LEARN TO PROGRAM SOUND EFFECTS

TRICKY TUTORIAL™ no. 10

```
87,34:POKE 207,99:NEXT
X+27,Y+88:SOUND 0,2,8,1
,12:FOR I = LEN(ABS(INT(
N GOSUB 50:READ BOOK:BOO
JZ = PEEK(242):LIST 800,1
```

EDUCATIONAL SOFTWARE

TABLE OF CONTENTS:

Load Instructions.....1
The Sound Effects Software.....2
Introduction.....3
The Train 5
The Graphics Effects Program..6
Menu1 21
Menu2 22
Fornext.....23
Poksound.....30
Sound16 32
Speaker.....35
Thatsall.....37
Appendix.....38

The SOUND EFFECTS Program Listings

Train.....55
Menu 1..... 58
Menu 2..... 60
Sound16.....63
Speaker..... 66

TRICKY TUTORIAL #10
SOUND EFFECTS

by Jerry White
with Becky Johnson

HOW TO LOAD

Tape....

With the BASIC cartridge in place, turn the computer on. Place the tape in your recorder, label side up. Make sure the tape is rewound and the counter is reset to zero. Push PLAY on the recorder, type CLOAD and press RETURN twice. If the program won't start to load, try positioning the tape forward or backwards a little. When the READY prompt appears, type RUN and press RETURN. We recommend you write down the number of your recorder's counter on the next page as each program example starts as this will make it easier to find each part later on.

Disk....

To LOAD and RUN the disk, first turn on your disk drive. When the busy light goes out, place the disk in the drive. With the BASIC cartridge in place, turn on the computer and the program will LOAD and RUN each part by itself.

ATARI is a registered trademark of Warner Communications, Inc.
Manual & Software © 1983 by Educational Software, Inc.

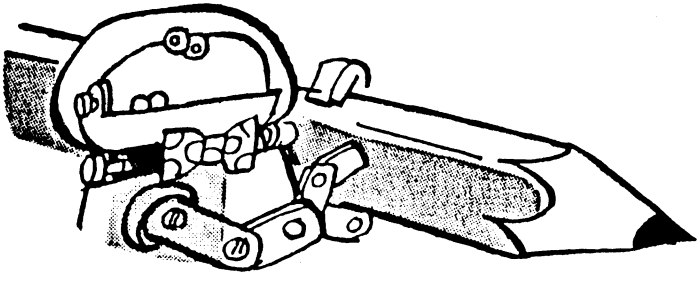
Any defective tapes or disks should be returned to:

Educational Software Inc.
4565 Cherryvale Ave.
Soquel, CA 95073

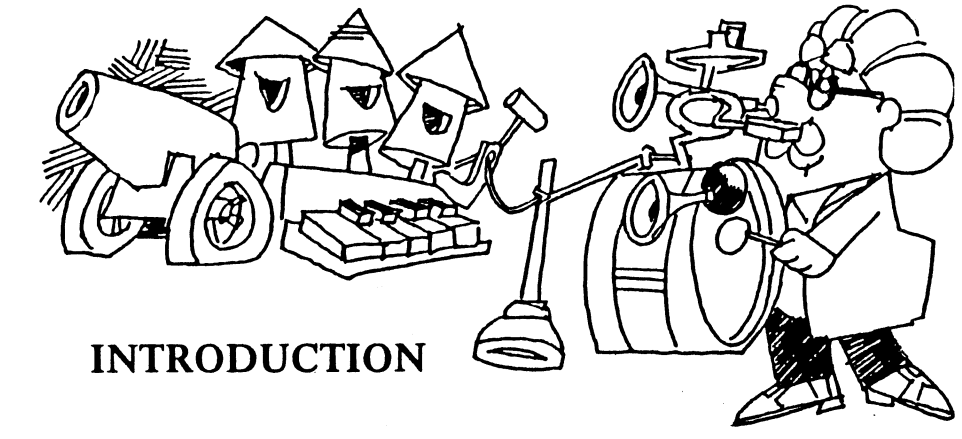
LOGO. SE

The Sound Effects Software

The following programs are included with this tutorial. At the top of each page in the manual is the name of the program which you should be running. Tape users should note the value of the counter as each program begins to load on the lines below. This will make it easier to find each program when you later run these examples.



PROGRAM	STARTS AT:
1 LOGO. SE	0
2 TRAIN	
3 EFFECTS	
4 MENU1	
5 MENU2	
6 FORNEXT	
7 POKSOUND	
8 SOUND16	
9 SPEAKER	
10 THATSALL	



INTRODUCTION

Dear Fellow Computerist,

In my long career working with computers I have often needed a 'ring—a—ding' and á 'root—a—toot' or two. In order to save a time I built up a nifty library of special sound effects. It occured to me that you might be in the same boat I was and could use my library too, as well as a few hints on how to create your own sound effects that 'speak' to your particular needs! The ATARI is one of the best 'clink—splat' sound generating computers on the market. It only seems reasonable to utilize all those marvelous capabilities. So, here they are; all my zoo bleats and varooooms in one Tricky Tutorial.

The only thing 'tricky' about this tutorial is how easy it is to use. I have actually provided you with several programs. First, it's ALL ABOARD for a train ride with PROTO. The next program is called EFFECTS . This program is designed to acquaint you with the technique of combining sound effects with graphics. If done properly a 'real time' synchronization between the sound effect and graphics display is possible, even in BASIC.

Then, it's on to MENU1 for a selection of special sound effects. You can listen to any one of them as often as you like. If you find one that you want to use in your own program it is an easy procedure to BREAK into the program and LIST that section of code to a list file. I'll explain more about that later. Once you have tried all the sound effects in MENU1 I will present you with MENU2. If that doesn't keep you busy I don't know what will! By the way, if there are small children in the house they will love these two programs.

LOGO.SE

Moving on, I offer you a nifty utility called FORNEXT. This program is engineered to allow you to vary time delays and FOR/NEXT loop variables to create your own unique sound effects. You can even save them for later use. I only wish I had such an easy time of it when I first started out!

After we have listened to some sound effects and used the FORNEXT utility it is time to get into the BITS and BYTES of what is going on in the ATARI. The next program, POKSOUND, will actually illustrate how to tinker with the SOUND REGISTERS to create sounds without using the SOUND statement in ATARI BASIC. I'll explain why we might want to do this and give you plenty of opportunitys to learn the HIGHS and LOWS of this technique.

This will lead us to really advanced 16 BIT sound generation that will allow greater frequency ranges than the SOUND statement of ATARI BASIC will. You can even create a piano keyboard using this technique.

Then, I'll show you how to use the speaker that is built into the ATARI to generate some interesting sounds. To end things, PROTO will dance to a familiar ditty called THATSALL. Once you learn how to make proper use of the ATARI sound features you'll really impress yourself with the creative things you'll be doing.

Have Fun,

Professor Von Chip

Professor Von Chip

THE TRAIN

The TRAIN is a fun program to show you how we might integrate sound effects and graphics. The code is very straight forward and utilizes a couple of nice assembly language routines to move PLAYER MISSILES on the screen. It also contains an assembly language time delay routine that uses internal timers. Feel free to "BREAK" into this program and steal whatever works for you.

The graphics have been restricted to the standard ATARI character set on a GRAPHICS 0 screen. Sound effects include a train whistle, the choo choo sound, PROTO's voice, and the pitter patter of PROTO's little feet. What PROTO also shows us is that the user must sometimes "have a little imagination".. So, without further delays, let's hop aboard the sound effects express. Have you got your ticket?

EFFECTS

THE GRAPHIC EFFECTS PROGRAM

EFFECTS demonstrates a wide range of sound effects and simple graphics. It requires no operating instructions for the first part. Just turn up the volume on your TV or monitor, watch, and listen. You can look at the code to see how graphics and sound are done together. The 2nd part of the program will appear on your display next. "ATARI BASIC SOUNDS" displays the code for several sound effects. Just hit the OPTION key to repeat the sound effect or the START key to continue.

There is a great deal that can be learned by studying the EFFECTS program listing. After having LOADed any of the programs into the computer's memory, you may use the LIST command to read the programmed instructions for that particular program. Each section of the program is labeled using inverse video REM statements. In this tutorial I will not try to explain the function of each BASIC command. Instead, I will briefly describe the routines rather than the individual instructions. If you don't know a particular BASIC command, consult the manual that came with your computer.

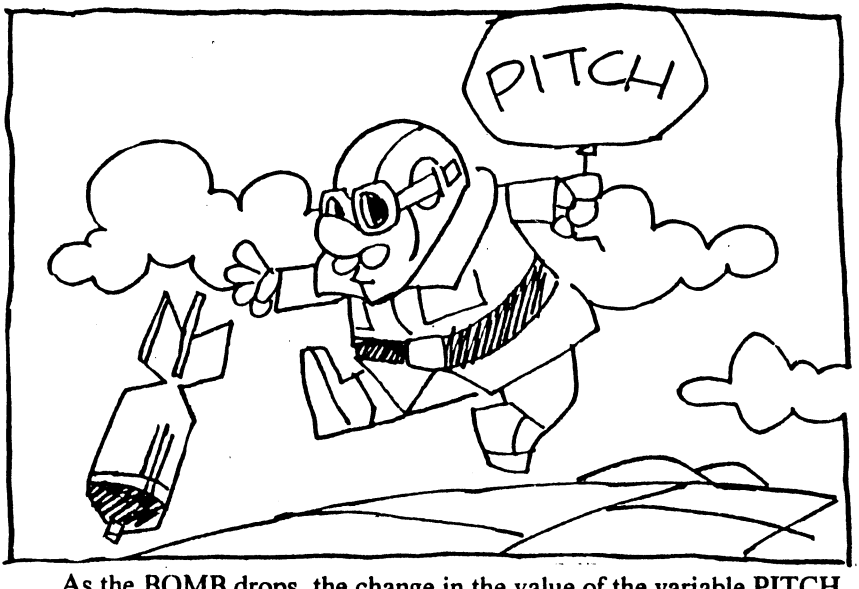
```
320 GOTO 350
324 REM
325 REM ***[illegible]***
326 REM
330 FOR PITCH=0 TO 240 STEP 5
    :SOUND 0,PITCH,0,14-PITCH/20
    :SOUND 1,PITCH,8,14-PITCH/20
    :NEXT PITCH
    :RETURN
340 GRAPHICS 0
    :SETCOLOR 1,0,8
    :SETCOLOR 2,0,0
    :SETCOLOR 4,0,0
    :POKE 752,1
    :RETURN
350 GOSUB 340
    :POSITION 15,21
    :? "TARGET"
```

EFFECTS

O.K., let's take a look at the EFFECTS program listing. Skipping over the author introduction, you will notice that line 320 says GOTO 350. This was done to skip over two subroutines. At line 350 we begin by using the subroutine at line 340. This simply sets up good old standard text mode, sets the colors to black and white, and turns off the cursor. RETURNing to line 350, we display the word "TARGET" near the bottom of the screen.

```
360 FOR PITCH=20 TO 174 STEP 2
370 POSITION 17,PITCH/10
    :? "B↓←O↓←M↓←B"
375 POSITION 17,PITCH/10
    :? "^↓←^↓←^↓←^"
380 SOUND 0,PITCH,10,2
    :NEXT PITCH
```

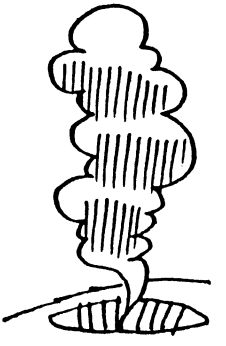
Lines 360 thru 380 contain a FOR/NEXT loop that will change the variable "PITCH" from 20 to 200, by increments of two. As the loop is executed, the word "BOMB" is displayed vertically on the screen. Four spaces are also displayed vertically. This is used to erase the word "BOMB" without erasing the entire screen.



As the BOMB drops, the change in the value of the variable PITCH will lower the frequency in the SOUND command in line 380.

EFFECTS

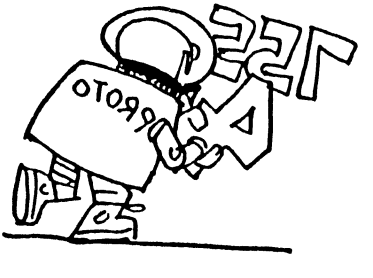
```
390 GRAPHICS 0
    :POKE 752,1
    :SETCOLOR 1,8,8
    :POSITION 15,21
    :? "T^^^^T"
    :POKE 755,4
400 POSITION 16,22
    :? "A^^E"
    :POSITION 17,23
    :? "RG";
```



At line 390, we erase the screen and begin repositioning the word "TARGET". Note the POKE 755,4 at the end of line 390. This creates what is called vertical reflect or mirror image. The word TARGET is spread over three lines of the screen, crudely simulating a hole left by the bomb.

```
410 FOR PITCH=20 TO 200 STEP 2
    :SOUND 0,PITCH,0,10-(PITCH/20)
420 SETCOLOR 2,4,10-(PITCH/20)
    :NEXT PITCH
430 POKE 755,2
    :FOR WAIT=1 TO 200
    :NEXT WAIT
```

The FOR/NEXT loop in lines 410 and 420 alters the PITCH and VOLUME of a SOUND command, and the luminance of a SETCOLOR command, creating an explosion sound and red flash on the screen. As the smoke clears, you will notice the word "TARGET" displayed in vertical reflect video. It will stay that way until the POKE 755,2 sets things back to normal in line 430. By the way, some interesting effects are demonstrated when a 1 or a 3 is poked into location 755 while inverse video characters are displayed on the screen.



Notice the WAIT loop in that same line. All it does is kill time. This is NOT the best way to do this. It is however, effective in this case and easily adjusted. The problem with FOR/NEXT loops is that they execute faster if they are near the beginning of a program. A more exact method of timing will be explained later on when we get to the section on timing sound effects in the appendix section of this manual.

If the FOR/NEXT loop has you a bit confused, don't despair. There is a program dedicated to the FOR/NEXT loop and a section of this manual called FORNEXT of all things, which will make this subject clear. After some experimentation using that program and possibly a bit of trial and error testing on your own, it will all make sense.

```
435 REM [illegible]
436 REM
440 GRAPHICS 21
    :SETCOLOR 0,1,14
    :COLOR 1
    :PLOT 5,5
    :DRAWTO 75,30
    :GOSUB 330
450 GRAPHICS 21
    :SETCOLOR 0,2,14
    :COLOR 1
    :PLOT 75,5
    :DRAWTO 5,30
    :GOSUB 330
460 GRAPHICS 23
    :SETCOLOR 0,3,14
    :COLOR 1
    :PLOT 5,5
    :DRAWTO 150,60
    :GOSUB 330
470 GRAPHICS 23
    :SETCOLOR 0,4,14
    :COLOR 1
    :PLOT 150,5
    :DRAWTO 5,60
    :GOSUB 330
480 GRAPHICS 0
    :SETCOLOR 2,6,0
    :POKE 752,1
484 REM
```

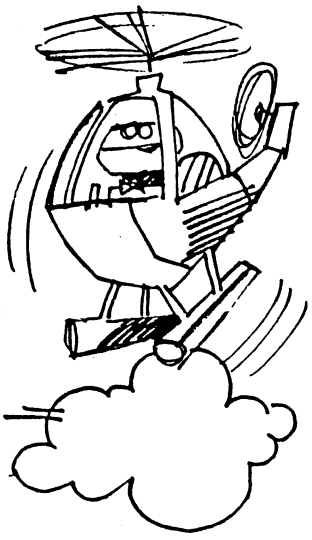
EFFECTS

The LIGHTNING routine uses GRAPHICS MODES 21 and 23. They are the same as modes 5 and 7, except they do not have a text window. Simply add 16 to the graphics mode # to get that mode without a text window. Lines are displayed on the screen to crudely simulate lightning. After each line is drawn, the subroutine at line 330 is executed. Why didn't we just put this routine in the next sequential line you ask? Subroutines are often used to save memory and typing. This subroutine is used four times. By using four GOSUB commands, we only need to enter the instructions found in line 330 once.

```
485 REM ***HELPLOPTER***
486 REM
490 DIM H1$(12),H2$(10),H3$(10),H4$(10),
    H5$(10),H6$(10),H7$(10)
500 H1$="~~~~~~~~~~"
510 H2$="~~~~~~~~~~"
520 H3$="~~~~~~~~~~"
530 H4$="~~~~~~~~~~"
540 H5$="~~~~~~~~~~"
550 H6$="~~~~~~~~~~"
560 H7$="~~~~~~~~~~"
570 SOUND 2,240,12,4
580 X=2
    :FOR Y=10 TO 0 STEP -1
    :POSITION X,10+Y
    :? "~~~~~~~~~~~"
    :GOSUB 230
590 POSITION X,5+Y
    :? H1$
    :POSITION X,6+Y
    :? H2$
    :POSITION X,7+Y
    :? H3$
    :POSITION X,8+Y
    :? H4$
    :GOSUB 230
600 POSITION X,9+Y
    :? H5$
    :GOSUB 230
    :POSITION X,5+Y
    :? H7$
    :POSITION X,6+Y
    :? H6$
```

EFFECTS

```
     :GOSUB 230
610 POSITION X,5+Y
     :? H1$
     :POSITION X,6+Y
     :? H2$
     :GOSUB 230
620 NEXT Y
630 FOR X=2 TO 28
     :GOSUB 230
640 POSITION X,5
     :? H1$
     :POSITION X,6
     :? H2$
     :POSITION X,7
     :? H3$
     :POSITION X,8
     :? H4$
     :GOSUB 230
650 POSITION X,9
     :? H5$
     :GOSUB 230
     :POSITION X,5
     :? H7$
     :POSITION X,6
     :? H6$
     :GOSUB 230
660 POSITION X,5
     :? H1$
     :POSITION X,6
     :? H2$
     :GOSUB 230
670 NEXT X
889 REM
```



Next we have our sputtering helicopter. Thanks to Dave Culbertson and C. E. Software for the use of their whirleybird.

In order to make the graphics easy to understand, we have defined seven strings. Again we use FOR/NEXT loops to alter the sound and the helicopter's appearance and position on the screen.

EFFECTS

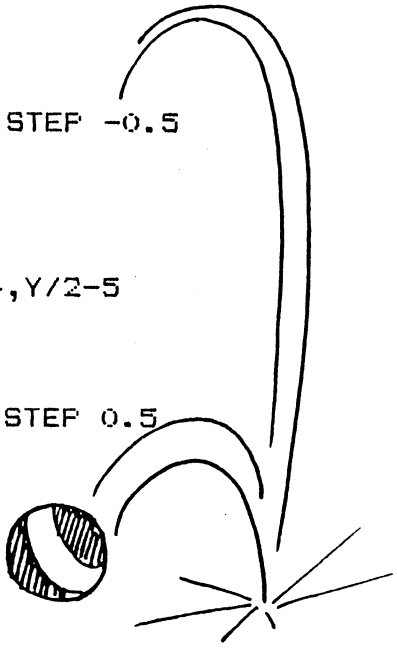
There are other ways to position character graphics on the screen. Since this is a tutorial on sound effects, I won't dwell on this or on the Player Missile Graphics routine. I would just like to point out one alternative method. The special function keys such as the arrows can be used within a string. They must be preceeded by pressing the ESC key when they are entered into a program. Once they are part of a string, they will be executed as if someone were actually holding the CTRL key while using the arrows to position the cursor.

```
890 REM ++++ALERT SIREN++++
891 REM
900 FOR ME=1 TO 6
    :FOR J=1 TO 20
    :SOUND 0,80+J,12,8
    :NEXT J
905 GRAPHICS 18
    :SETCOLOR 0,ME,8
906 POSITION 7,4
    :? #6;"ALERT"
910 SOUND 0,80,10,12
    :SOUND 1,100,10,12
    :SOUND 2,13,4,12
920 FOR W=1 TO 100
    :NEXT W
    :GOSUB 250
    :NEXT ME
940 FOR V=12 TO 0 STEP -0.1
    :SOUND 0,(20-V)*10,10,V
    :SOUND 1,(20-V)*10+20,10,V
    :SOUND 2,13,4,V
    :NEXT V
1000 REM
```

By the sound of that helicopter, it's probably crashed by now so let's go on to the ALERT SIREN routine. There's no animation here, the nested FOR/NEXT loops change only the sound and color of the word ALERT. NESTED??? Nested refers to one or more loops within another. Again, we will discuss this in depth when we look into the FORNEXT program.

EFFECTS

```
1001 REM ***BOUNCING BALL***
1002 REM
1005 ? "T"
     :SETCOLOR 2,4,2
     :POKE 752,1
1006 FOR ME=1 TO 3
1010 FOR Y=22 TO 10 STEP -0.5
1020 POSITION 18,Y+1
     :? "~"
     :POSITION 18,Y
     :? "●"
1030 SOUND 0,Y*10,14,Y/2-5
     :NEXT Y
1035 FOR Y=1 TO 9
     :NEXT Y
1040 FOR Y=10 TO 22 STEP 0.5
     :POSITION 18,Y-1
     :? "~"
     :POSITION 18,Y
     :? "●"
     :NEXT Y
     :NEXT ME
1049 REM
```

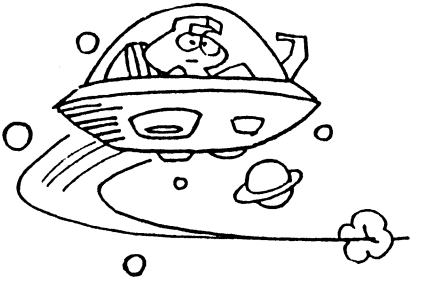


The BOUNCING BALL uses the ATARI ball character (CTRL T) to demonstrate simple animation. As the ball rises, the sound increases in pitch and decreases in volume. This combination provides what I call a "BOING" effect.

```
1050 REM ***PONG-LIKE SHOOTERS***
1051 REM
1100 GRAPHICS 0
     :POKE 752,1
     :? "~"
.110 SETCOLOR 2,0,0
     :X=20
     :Y=80
(120 POSITION 5,3
     :? "."
     :POSITION 27,6
     :? "."
```

EFFECTS

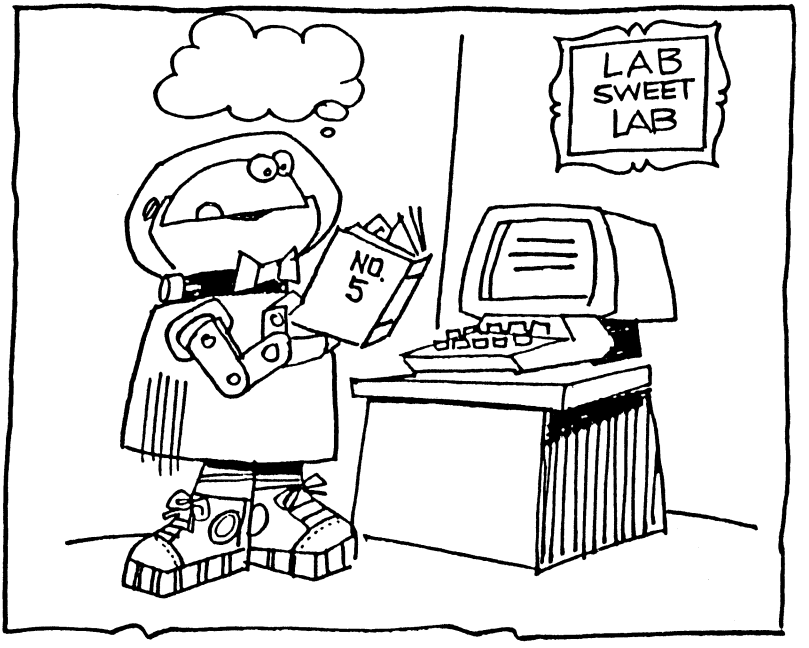
```
1130 FOR ME=1 TO 10
     :RX=RND(0)*37+1
     :RY=RND(0)*21+1
     :POSITION RX,RY
     :? "."
     :NEXT ME
1140 A=PEEK(106)-8
     :POKE 54279,A
     :PMBASE=256*A
1150 POKE 559,46
     :POKE 53277,3
     :POKE 53248,X
     :POKE 53256,3
1170 FOR I=PMBASE+512 TO PMBASE+640
     :POKE I,0
     :NEXT I
     :POKE 704,100
1180 FOR I=PMBASE+512+Y TO PMBASE+516+Y
     :READ A
     :POKE I,A
     :NEXT I
1190 DATA 60,126,255,126,60
1200 SOUND 0,13,4,2
1210 FOR X=20 TO 80 STEP 2
     :POKE 53248,X
1220 FOR I=0 TO 6
     :POKE PMBASE+511+Y+I,
      PEEK(PMBASE+512+Y+I)
     :NEXT I
     :Y=Y-1
     :NEXT X
1230 SOUND 0,13,4,10
     :FOR X=82 TO 220
     :POKE 53248,X
     :NEXT X
     :POKE 704,20
1240 FOR V=10 TO 0 STEP -0.5
     :SOUND 0,13,4,V
     :NEXT V
1250 FOR ME=0 TO 10
```



```
     :SOUND 1,0,2,ME+2
     :FOR I=0 TO 6
     :POKE PMBASE+511+Y+I,PEEK
      (PMBASE+512+Y+I)
     :NEXT I
1260 SOUND 1,0,0,0
     :Y=Y-1
     :NEXT ME
     :POKE 53248,220
1270 SOUND 0,25,4,2
     :POKE 53256,1
     :FOR X=220 TO 0 STEP -1
     :POKE 53248,X
     :NEXT X
     :SOUND 0,0,0,0
     :POKE 53248,120
1280 FOR ME=15 TO 0 STEP -1
     :POKE 53277,3
     :POKE 53256,3
     :FOR P=0 TO 10
     :SOUND 0,P,2,ME
     :SOUND 1,P,2,ME
1290 NEXT P
     :POKE 704,ME*16+6
     :POKE 53277,1
     :RX=RND(0)*37+1
     :POSITION RX,22
     :? "."
     :?
     :NEXT ME
1700 GOSUB 250
     :? "T"
     :POKE 752,1
     :?
1749 REM
```

EFFECTS

This brings us to a very complex subject call Player/Missile (P/M) graphics. Again I will refrain from explaining this subject since it requires and has a separate tutorial of it's own. In case you're interested, it's called Tricky Tutorial #5: Player/Missile Graphics.



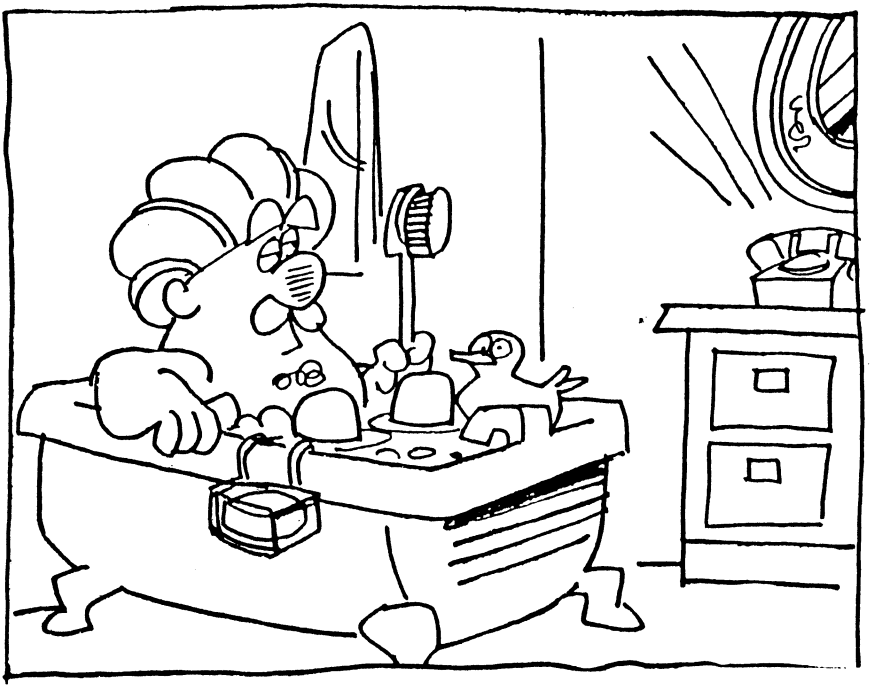

```
1750 REM [TELEPHONE RINGING]
1751 REM
1800 JIFFY=60
     :FOR R=1 TO 2
     :FOR ME=1 TO 20
     :SOUND 0,25,10,1
     :SOUND 1,0,2,1
     :FOR J=1 TO 2
     :SOUND 0,25,10,8
1805 SOUND 1,0,2,8
     :NEXT J
     :SOUND 0,0,0,0
     :SOUND 1,0,0,0
     :SETCOLOR 2,ME,4
```

```
     :NEXT ME
1810 FOR V=7 TO 0 STEP -0.1
     :SOUND 0,25,10,V
     :SOUND 1,0,2,V
     :NEXT V
     :GOSUB 8200
     :NEXT R
1820 L$="PARDON~ME~WHILE~I~ANSWER~THE~
      PHONE"
     :?
     :?
     :FOR J=1 TO LEN(L$)
     :? L$(J,J);
     :GOSUB 230
     :NEXT J
1830 FOR ME=1 TO 10
     :SOUND 0,25,10,2
     :SOUND 1,0,2,1
     :FOR J=1 TO 2
     :SOUND 0,25,10,8
     :SOUND 1,0,2,8
1840 NEXT J
     :NEXT ME
     :GOSUB 250
     :GOSUB 8200
1842 L$="HELLO"
     :POSITION 15,8
     :FOR J=1 TO LEN(L$)
     :? L$(J,J);
     :GOSUB 230
     :NEXT J
1850 GOSUB 8000
     :POSITION 12,12
     :L$="IT'S~FOR~YOU."
     :FOR J=1 TO LEN(L$)
     :? L$(J,J);
     :GOSUB 230
     :NEXT J
     :GOSUB 8200
2100 FOR X=1 TO 24
     :?
```

EFFECTS

```
     :SOUND 0,X,2,2
     :SOUND 1,X,4,2
     :SOUND 2,X,6,2
     :SOUND 3,X,12,6
2110 FOR Y=1 TO 5
     :NEXT Y
     :NEXT X
     :GOSUB 250
2200 FOR X=14 TO 0 STEP -0.5
     :SOUND 0,91,12,X
     :NEXT X
```

The TELEPHONE RINGING routine demonstrates that sound effect, plus the sound of a teletype machine. These results are achieved using what I call short sounds. Identical SOUND commands can be made to sound different when the duration of the sound is limited.



By now you may be wondering how to select the pitch, distortion, volume, and duration, to create other sound effects. I know of only one way. You must experiment and use the old trial and error method.

EFFECTS

```
2270 REM ****************************
2271 REM *                          *
2272 REM ****************************
2273 GRAPHICS 0
     :POKE 752,1
     :?
     :?
     :?
2274 ? "~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
2275 ? "~~~~~~~~~~~~~~~~~~~/~~~~~~~~"
2276 ? "~~~~~ ┌──────────┐~~~~~~~~~"
2277 ? "~~~~~ └──────────┘~~~~~~~~~"
2278 ? "~~~~~~|~~~~~~~~/~ |~~~~~~~~"
2279 ? "~~~~~~|~~~~~~~/~~ |~~~~~~~~"
2280 ? "~~~~~~|~~~~~~/~~~ |~~~~~~~~"
2281 ? "~~~~~~|~~~~~/~~~~ |~~~~~~~~"
2282 ? "~~~~~~|~~~~/~~~~~ |~~~~~~~~"
2283 ? "~~~~~~|~~~/~~~~~~ |~~~~~~~~"
2284 ? "~~~~~~|~~/~~~~~~~ |~~~~~~~~"
2285 ? "~~~~~~|~/~~~~~~~~ |~~~~~~~~"
2286 ? "~~~~~~|/~~~~~~~~~ |~~~~~~~~"
2287 ? "~~~~~~└──────────┘~~~~~~~~"
2288 ? "~~~~~~~~~~|~~~~|~~~~~~~~~~"
2289 ? "~~~~~~~~~~|~~~~|~~~~~~~~~~"
2290 ? "~~~~~~~ ┌──┘  └──┐~~~~~~~~"
2291 ? "~~~~~ ┌──┘      └──┐~~~~~~"
2292 ? "~~~~~ └──────────┘~~~~~~~~"
2294 L$="                          "
2295 FOR R=15 TO 7 STEP -1
2296 FOR L=1 TO 21 STEP 10
2297 POSITION 8,R
     :? L$(L,L+9)
2298 SOUND 0,L+3*R,10,10
2299 FOR W=1 TO 20
     :NEXT W
2300 NEXT L
     :NEXT R
     :SOUND 0,0,0,0
2301 FOR W=1 TO 100
     :NEXT W
2303 POKE 752,0
```

EFFECTS

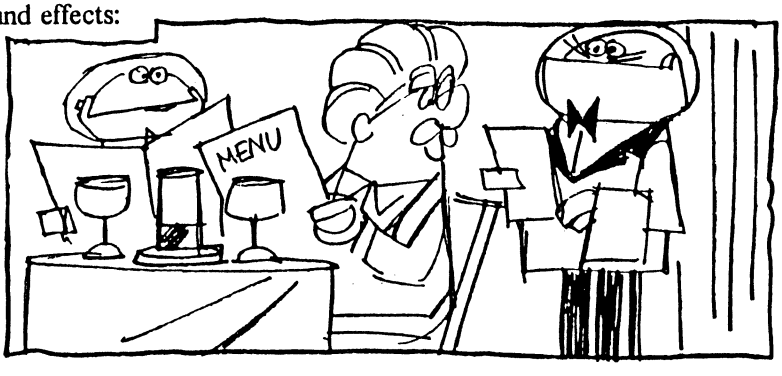
The next section, entitled FILL first prints the image of a goblet using standard ATARI character graphics. Then, as the sound of a glass being filled is created, the proper substring of L$ is printed to simulate the glass being filled.

```
2320 REM ********DISPLAY FOR NEXT LOOPS********
2340 REM
```

To provide more food for thought, we now come to the section labeled, "DISPLAY FOR NEXT LOOPS". This is a series of nested FOR/NEXT loops that shows the BASIC instructions on the screen as these instructions are executed. Simply listen to the sound effect and study the code which produced it as long as you'd like to. When you wish to go on, just press any key to continue.

MENU1

This is really the heart of this tutorial.... two separate programs containing 36 separate sound effects that are easy for you to access, alter, copy, and use. The first program is MENU1 and contains the following sound effects:



SELECT	SOUND EFFECT	LINE#
A	YOU WIN	1100
B	YOU LOSE	1200
C	GUESS WHAT	1300
D	WINNER	1400
E	HORN1	1500
F	WHAM	1600
G	DOOR SLAM	1700
H	ELECTRIC BELL	1800
I	SLIDING DOWN	1900
J	FALLING	2000
K	HAPPY	2100
L	PING–PONG	2200
M	EXPLOSION	2300
N	BEEP–BEEP	2400
O	DEE DUMP	2500
P	HORN2	2600
Q	BUZZER	2700
R	QUIT	

MENU 2

MENU2

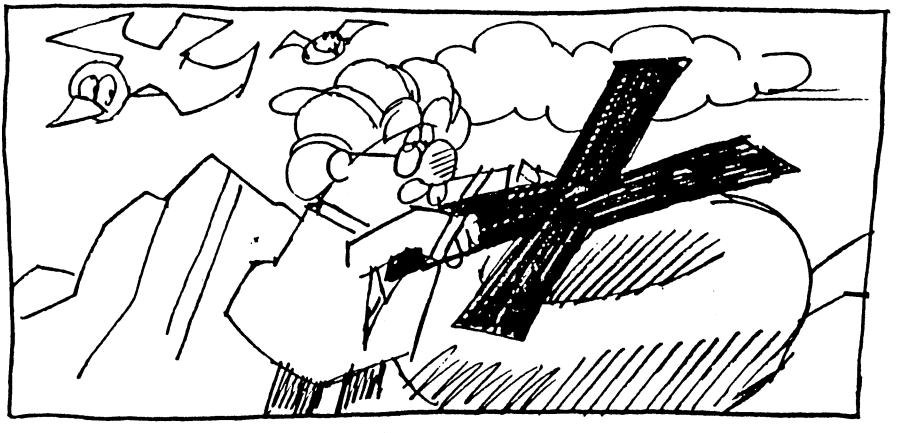
MENU2 contains these sound effects:

SELECT	SOUND EFFECT	LINE#
A	BRONX CHEER	100
B	LOCOMOTIVE	200
C	TWINKLE	300
D	SLIDE DOWN	400
E	CRICKET	500
F	DOOR SLAM	600
G	RACING CAR	700
H	HORSE NEIGH	800
I	BUZZER	900
J	SOFT BELL	1000
K	ROCKET BOOSTER	1100
L	HEART BEAT	1200
M	THUNDER	1300
N	BIRD CHIRP	1400
O	WATER DRIP	1500
P	CARD SHUFFLE	1600
Q	OCEAN WAVES	1700
R	TELEPHONE	1800
S	RAINSTORM	1900
T	QUIT	

While some of the sound effects titles are the same on both menus they are not the same sound effects.

FORNEXT

This program is actually a rather basic utility that allows you to create and save your own sound effects. It is based on a simple "nested" FOR/NEXT loop that controls pitch and volume for a single sound voice. WOW! That covers a lot of terms. It is important to understand them all, so, let's start with the "nested" FOR/NEXT loop.



Just what is a "nested" FOR/NEXT loop? I assure you it has nothing to do with birds, although, there are times you may think it is for the birds! To understand "nested" FOR/NEXT loops you must understand a simple FOR/NEXT loop such as this:

```
1010 ? "THIS IS ";
1020 FOR J= 0 TO 3
1030 ? STR$(J);", ";
1040 NEXT J
1050 ?
```

It shouldn't take much study to see that this code will produce the following RUN:

THIS IS 0,1,2,3

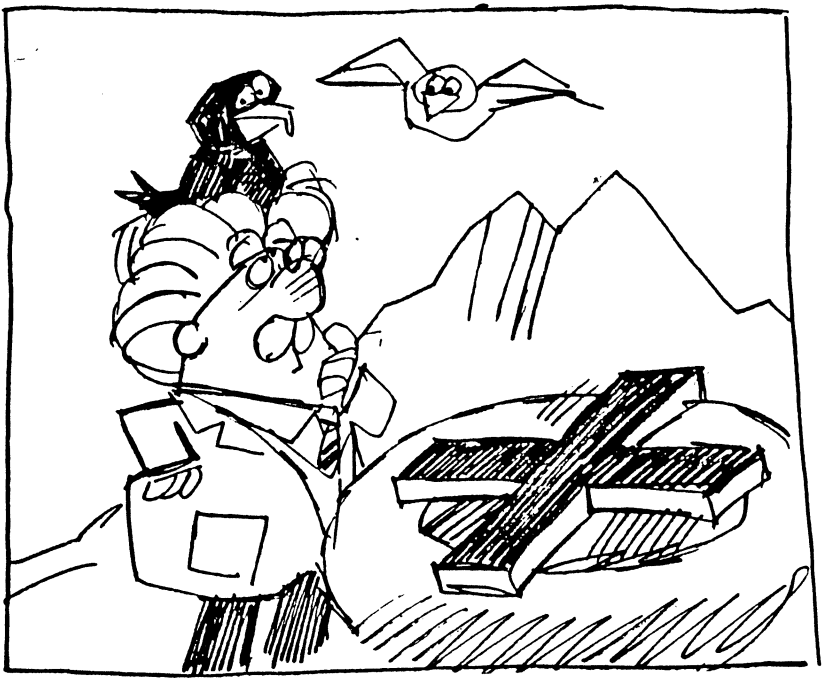
FORNEXT

I can change the results of this simple FOR/NEXT loop by "nesting" it inside another loop. For example:

```
1000 FOR I=110 TO 40 STEP 10
1010 ? "THIS IS ";
1020 FOR J=10 TO 3
1030 ? STR$(I);STR$(J);", ";
1040 NEXT J
1050 ?
1060 NEXT I
```

The RUN would look like this:

```
THIS IS 100, 101, 102, 103
THIS IS 200, 201, 202, 203
THIS IS 300, 301, 302, 303
THIS IS 400, 401, 402, 403
```



FORNEXT

I was able to generate an output containing numbers that were equal to or greater than 100 without using multiplication or any numbers greater than 40. This particular example does not have much practical application other than demonstrating the concept of "nested" FOR/NEXT loops. Before we continue this discussion let's take a look at the SOUND statement found in BASIC.

SOUND 0,40,10,12

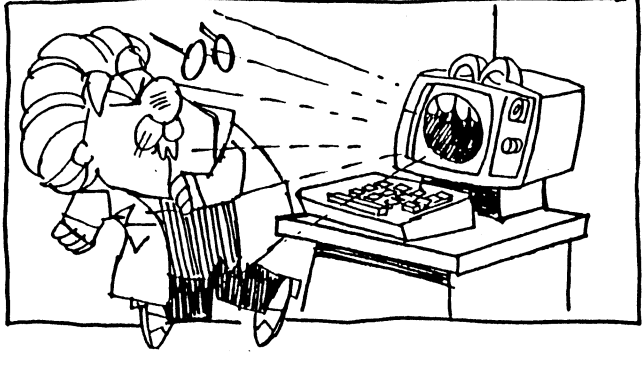
volume
distortion
pitch
voice

VOICE: is one of four sound voices in ATARI BASIC. They are numbered zero (0) through three (3). Using the SOUND statement would allow you to turn on each voice simultaneously with different characteristics.

PITCH: would probably better be described as frequency. A siren has a high pitch and a heartbeat has a low pitch. The PITCH range is from 0 to 255 with 1 being the highest PITCH and 255 being the lowest PITCH.

DISTORTION: is variable from 0 to 14 in steps of 2 (i.e. 2,4,6,8,10,12,14). It is the distortion parameter that allows you to create some really "spacey" sounds. A DISTORTION of 10 is a pure tone.

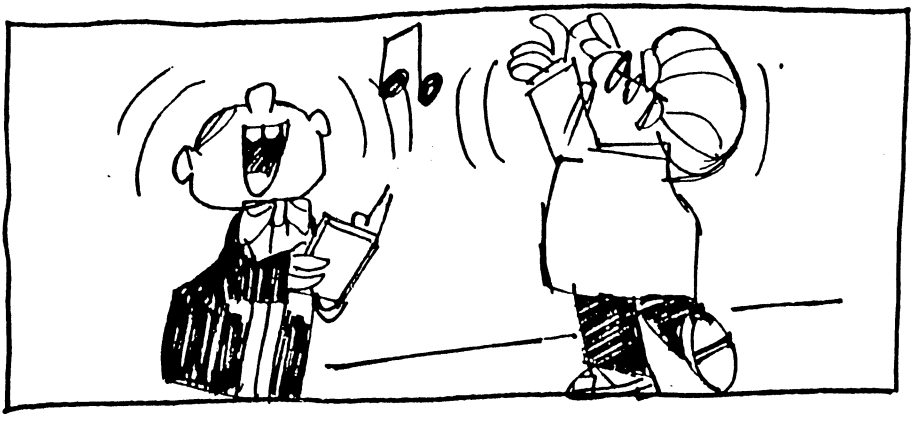
VOLUME: is also variable from 0 to 14 in steps of 2 and is just like the volume control on your TV. It determines how loud the sound will be. The loudest volume is determined by the highest setting of the volume control on your TV set.



FORNEXT

Now then, using a simple FOR/NEXT loop I can listen to the entire PITCH range of a single sound VOICE:

```
1000 FOR I=0 TO 255
1010 SOUND 0,I,10,10
1020 NEXT I
```



This will start at a very high PITCH and end in a very low PITCH. If this example is run as a program by itself then the sound channel will turn OFF when it is done. However, if lines 1000 through 1020 were "inside" a larger program the sound would remain ON, constantly sounding the last PITCH (255) used in the loop. To insure the sound VOICE were turned OFF before executing any further code I would have to insert:

```
1030 SOUND 0,0,0,0
```

If I wanted to hear the effect in the opposite direction I would simply change line 1000 to read:

```
1000 FOR I= 255 TO 0 STEP -1
```

Now it is not necessary to turn OFF the sound VOICE because the last PITCH value executed was zero (0). Also, if I intend to immediately execute another SOUND loop utilizing the same VOICE, I need not worry about turning the VOICE OFF after the first loop. Paying attention to these details can save memory by omitting code (line 1030) that is not needed.

FORNEXT

I can now write a "nested" FOR/NEXT loop to listen to the effects of DISTORTION on different PITCHes. For example:

```
1000 FOR P=10 TO 100 STEP 10
1010 FOR D=2 TO 14 STEP 2
1020 SOUND 0,P,D,10
1030 FOR W=1 TO 200:NEXT W
1040 NEXT D
1050 NEXT P
```

NOTE: PLEASE READ THE APPENDIX FOR TIPS ON PROGRAMMING TECHNIQUE THAT PERTAIN TO THIS EXAMPLE.

This example demonstrates FOR/NEXT loops "nested" three (3) deep. The inner most loop is a simple time delay to give us time to hear each effect. The next loop uses the loop variable "D" for DISTORTION and simply samples each successive level of DISTORTION for a given PITCH. The PITCH is controlled by the outer most loop using the loop variable "P" for PITCH.

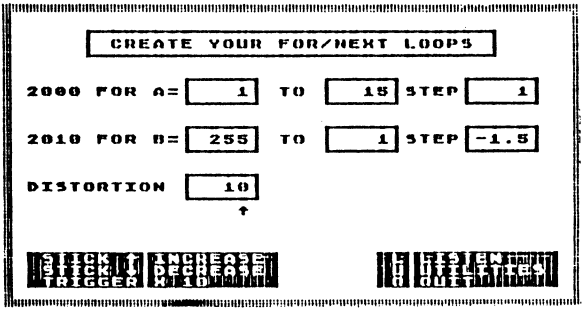
A variation of this example would be:

```
1000 FOR V=1 TO 10
1010 FOR P=1 TO 10
1020 SOUND O,P,10,V
1030 NEXT P
1040 NEXT V
```

This will sound something like a bird chirping. You can write this yourself - OR - you can use our handy dandy program called FORNEXT (what else!) to try this. If you don't quite like the sound you will be able to change the variables within the loops to get the effect you want. Then you can LIST it to an ATASCI file on disk or tape to be ENTERED into whatever program you like later on

FORNEXT

The screen for FORNEXT should look like this:



Plug a joystick into port one. Moving the stick left or right will move the index arrow to one of four positions across the top of the screen.

1. The first position points to the beginning of the program statement. When you are in this position you are able to move the index arrow up or down to a new line of options on the screen by moving the joystick forward or backward. Once you move the index arrow horizontally away from the first position moving the joystick forward or backward has a different function.

2. In the second position you will be pointing to the FROM variable for the VOLUME loop. Moving the joystick forward will increment the value and moving the joystick backward will decrement the value. If you hold down the button on the joystick while increasing or decreasing the value it will change in increments of 10 or −10 accordingly.

3. The third position is the TO variable and follows the same rules outlined in #2.

4. In the fourth position you can vary the STEP variable from −15 to 15. If you are going FROM a high number TO a low number you must STEP by some NEGATIVE (−) value, otherwise, the loop will never exceed a single execution. On the other hand, if you are going FROM a small number TO a larger number then your STEP increment must be a POSITIVE value. If you have a zero (0) in the STEP value the loop will be locked up.

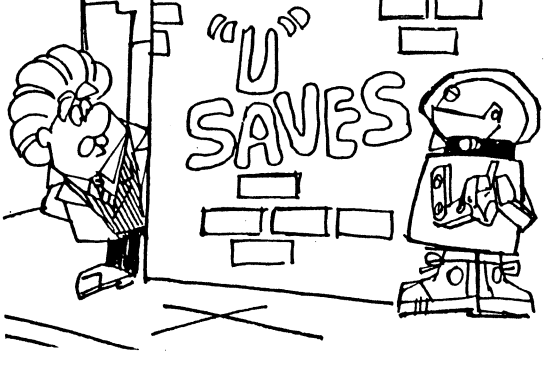
When a STEP of 1 is used in any loop the STEP portion of the FOR − TO statement is not necessary. It is here for purposes of demonstration but will not occur on any sound effects "saved" by this utility.

Moving the index arrow to the second line acts the same as on the first line. The only difference is on the allowable range of values. In the first line (VOLUME) the range is 0 to 15 and on the second line (PITCH) the range is 0 to 255. The third line only contains one variable (DISTORTION) and will only allow values of 0 to 14 in steps of 2 (i.e. 2,4,6,8,10,12,14).

You can use a STEP of .5 or .25 in either of the loops even though the actual data stuffed into the sound registers by the SOUND statement are be INTEGER values. The effect of using fractional STEP values is that a time delay is generated for that particular loop. It just keeps STEPPING by fractions until it reaches a new INTEGER value that will effect a change in the sound registers via the SOUND statement.

Once you enter values you want to try just push "L" on the keyboard (for LISTEN) and you will hear your sound effect, provided, of course, you have the volume on your TV set turned up (a minor detail indeed!). If you want to "save" your sound effect just push "U" on the keyboard (for UTILITIES) and you will see your code listed to the screen and be given several options, one of which is to save the routine. It will allow LISTing the file to disk or tape. Also, you may listen to it as before, but, this time you have the option to turn off all interrupts to the 6502 microprocessor (i.e. screen display) and hear the difference in timing.

FORNEXT is somewhat limited as it does not provide sophisticated timing controls or a third loop to control DISTORTION. Nor does it allow for more than one VOICE. However, you can create very different effects by just putting two of these simple loops back to back with PITCH changes or variations in the STEP values. Remember, "SAVEing" two loops to LISTed files will produce code with identical line numbers so be sure to RENUMBER the segments properly.



POKSOUND

I have mentioned "SOUND REGISTERS" a couple of times and I am sure you want to know what those varmints are. First, let me list them:

LABEL	DECIMAL	HEX	DESCRIPTION
AUDF1	53760	D200	Audio Channel 1 Pitch
AUDC1	53761	D201	Audio Channel 1 Control
AUDF2	53762	D202	Audio Channel 1 Pitch
AUDC2	53763	D203	Audio Channel 1 Control
AUDF3	53764	D204	Audio Channel 1 Pitch
AUDC3	53765	D205	Audio Channel 1 Control
AUDF4	53766	D206	Audio Channel 1 Pitch
AUDC4	53767	D207	Audio Channel 1 Control
AUDCTL	53768	D208	Audio Control

Excluding the last one, there are four (4) pairs of memory locations, called registers, for each sound channel. The first of any pair is the PITCH control. That is simple and straight forward. The contents of that register relates directly with the PITCH variable in the SOUND statement. For example:

```
SOUND 0,120,10,8
```

will put the value 120 into the AUDF1 register.

The second of the pair of registers combines the DISTORTION and VOLUME variables. The formula for that combination is:

```
VOLUME + 16 * DISTORTION
```

That is, it takes the VOLUME variable and adds it to 16 times the DISTORTION variable. Again, from our example:

SOUND 0,120,10,8

we will find 8+16*10, or 168, in the AUDC1 location. If you try to PEEK at these locations to see what is there you will be dumbfounded. These memory locations are WRITE ONLY because the operating system is constantly fooling around with them. Your original values were snatched away to configure the audio signal and then promptly trashed so don't distress yourself trying to figure out what is happening by PEEKing at the audio control registers.

So, uh, ok! What's the advantage when I POKE the sound registers as opposed to using the SOUND statement? Well, not much really. Aside from utilizing 16 bit sound capabilities, which we'll talk about later, there could conceivably be some savings in time. The SOUND statement must do several things. If you knew exactly what values to POKE into a PITCH register without having to compute them in your program then you could save whatever time is lost when the SOUND statement performs the math on the DISTORTION and VOLUME variables before it stuffs the appropriate AUDIO CONTROL register. Since most sound effects involve delays or integrate with some sort of graphics this advantage would really be negligible.

If you run POKSOUND you will have the opportunity to see the effect when you directly POKE the sound registers. Simply install a joystick, and move the stick forward or backward to select the register you want to POKE. Once your selection is made just push the trigger button on the joystick. Now you can move the stick forward or backward to increase or decrease the value being POKEd into the memory location. When you are satisfied with your value push the button again and make another selection. Pressing the SPACE BAR will zero all the values. Press ESC to exit.

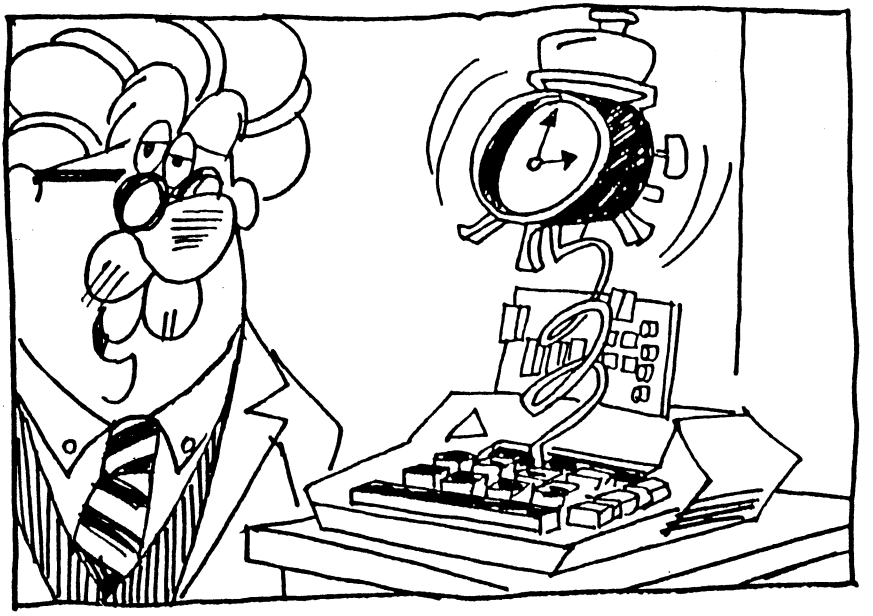
This program will allow you to listen to several sound channels at once. Try playing with DISTORTION and PITCH in different combinations. If you are musically inclined you can create chords this way.

More thorough and detailed technical explanations concerning the sound registers can be found in DE RE ATARI.

SOUND16

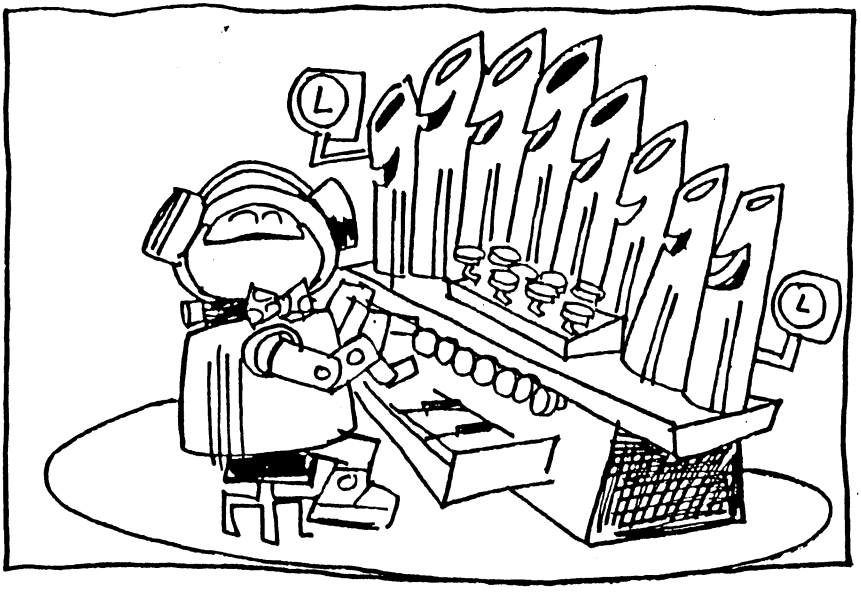
That just wasn't enough.... you want more! Okee Dokee! How about combining registers to create 16 bit sound. Until now you have been putting a value from zero (0) to 255 in the PITCH register. Are you wondering what this number was used for? I'll tell you anyway!

The computer has what is called an internal CLOCK. This produces several tiny pulses of electrical energy every second. The rate of these pulses is called the SYSTEM CLOCK FREQUENCY. It is sort of like the clock a human might use (you remember humans). Someone might give you a task to perform and give you exactly five seconds to do it. If you fail then you get fired and someone who can do it in five seconds replaces you. Well, we sort of do the same thing to the 6502 MICROPROCESSOR which is the heart of the ATARI. We're a little harder on the 6502 though. Instead of dealing in seconds we deal in little biddy things called nanoseconds. A nanosecond is one one millionth of a second. The 6502 is sort of lazy working at 1, 2, or 3 MegaHerz (that's millions of pulses per second).



SOUND16

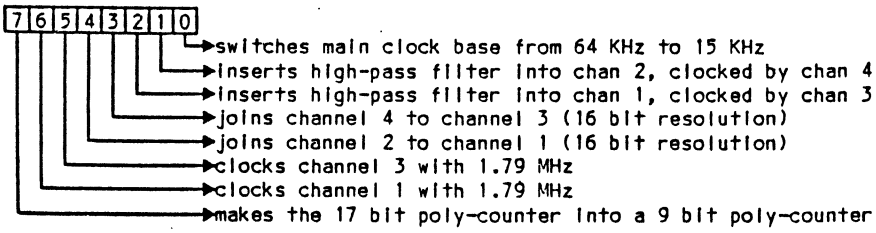
Anyway, we use that internal clock frequency to generate the frequencies audible to the human ear by dividing the SYSTEM CLOCK FREQUENCY with the PITCH value. If we use a bigger number, then we can get more frequencies. Thus, in their wisdom, ATARI provided a means by which AUDIO registers can be combined to form 16 bit registers and produce greater sound ranges. This was primarily done to satisfy the demands of music synthesis which needs more than four octaves to work with. The drawback is that you lose a VOICE each time you combine two registers. You can have four lousy singers or two terrific singers! Actually, you can also have two lousy singers and one terrific singer. For most applications four VOICES is best. Only very serious programmers concerned with music synthesizers will concern themselves with 16 bit sound and I just know there are one or two of you out there.



The first thing you should understand is that you can no longer use the SOUND statement when performing 16 bit sound. The SOUND statement would "confuse" the registers since they are paired and the SOUND statement only affects ONE register.

SOUND16

The "pairing" of sound registers is accomplished through the use of the AUDCTL register located at memory location 53768. Depending on its contents (i.e. which BITS are set) it will perform the following functions:


AUDCTL ($D208 = 53768)
If set, this bit...
7 6 5 4 3 2 1 0
switches main clock base from 64 KHz to 15 KHz
inserts high-pass filter into chan 2, clocked by chan 4
inserts high-pass filter into chan 1, clocked by chan 3
joins channel 4 to channel 3 (16 bit resolution)
joins channel 2 to channel 1 (16 bit resolution)
clocks channel 3 with 1.79 MHz
clocks channel 1 with 1.79 MHz
makes the 17 bit poly-counter into a 9 bit poly-counter

AUDCTL bits 3 and 4 allow two channels to be joined together for 16 BIT resolution. If BIT 3 were set then channel 4 would become the HIGH ORDER BYTE and channel 3 would become the LOW ORDER BYTE of the pair. You can hear the effect with the following program:

```
2000 SOUND 0,0,0,0
2010 POKE 53768,80
2020 POKE 53763,166:POKE 53761,160
2030 FOR COURSE=0 TO 255
2040 POKE 53762,COURSE
2050 FOR FINE=0 TO 255
2060 POKE 53760,FINE
2070 NEXT FINE
2080 NEXT COURSE
```

This is just a small example of 16 bit sound. I am having PROTO put together an advanced sound tutorial that will explain all this in great detail.

SOUND16 demonstrates 16 bit sound. The first part of SOUND16 will simply play musical notes while displaying them on the screen for your convenience. The second part of the program will allow you to use a joystick to change the values of the sound registers to hear the effect when they are "paired" for 16 bit sound.

SPEAKER

The SPEAKER program allows you to see how an assembly language routine can create some interesting sound effects utilizing the speaker built into the ATARI. Using the joystick you can vary several parameters of the routine. Once a number has been selected it is moved directly into the assembly language routine by equating the proper character in the assembly language STRING to the CHR$ of the value.

This is one way to pass a variable to an assembly language routine. The most common way is to pass the variable through the USR statement. This requires some extra overhead in writing the assembly language routine to arrest the variable and maintain the STACK. In the case of SPEAKER we must know the exact location of each variable in the STRING representing the assembly language routine. Then we only need to convert the variable value to its ATASCII character representation and insert it into the string. For example suppose we have a routine in ASY$ which is 20 characters in length. Now, we know that the 16th character is a variable we may want to change.

SPEAKER

Suppose we want to make this variable the number 66. We could do the following:

```
2030 A$=CHR$ (66)
2040 ASY$(16, 16) =A$
```

or, simply:

```
2030 ASY$(16,16) =CHR$(66)
```

Of course there is one very important restriction to passing variables to an assembly language routine in this way.... you cannot use a value exceeding 255 because that is the limit of a single BYTE.

SPEAKER can be useful for certain kinds of warnings and audible cues to a user, however, it is rather abrasive. I know I have been tempted to send PROTO on search and destroy missions to software vendors who like to use the speaker often in their programs.

THATSALL

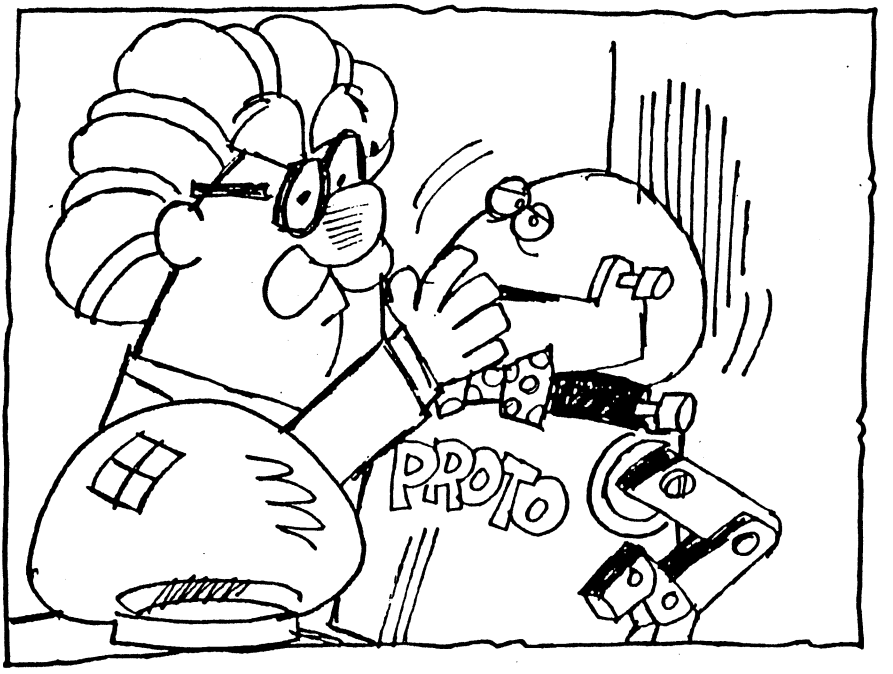
Finally, we come to THATSALL where PROTO will show off a little. I wanted you to see how to make some simple MUSIC using a single sound channel. If you are really interested in creating music on the ATARI then try our TRICKY TUTORIAL # 6, Sound and Music.

APPENDIX

A FEW IMPORTANT NOTES

POLICY

We have a very "open" policy regarding the TRICKY TUTORIALS at Educational Software Inc. This software is designed to teach you fun and interesting programming techniques in an easy to learn way. We also believe that "learning by example" is easier than a hit-and-miss approach. Therefore, we do not secure any of our code and encourage the user to look into the programming examples we provide to see just how we did it. Futhermore, you can "strip" out sections of our code, modify it, and create code of your own that is a variation of the original.



APPENDIX

TURNING OFF THE SOUND

Occasionally, in accordance with our policy, you may choose to BREAK one of our programs while it is running so you can peek at the code. If you do perform a BREAK while the computer is sounding off it will not shut off the sound channels. Because the code that causes variations in the sound channels that create the special sound effect you were listening to is "stopped" the sound continues to whine away at whatever control values were in the registers when you performed the BREAK. This could be extremely irritating while you try to study the code. There are a couple of ways to muffle the beast.

You could hit the SYSTEM RESET key, but, that is rather drastic. The computer has to do all sorts of internal things when the SYSTEM RESET key is pressed and you really only want to shut off the sound.

As you know, you can enter commands and statements directly into the computer while in BASIC, without a line number, and they will execute right away. There is a name for this feature.... it is called "EXECUTE IMMEDIATE". Seems reasonable, right? Anyway, you could type in:

```
FOR I=0 TO 3:SOUND I,0,0,0:NEXT I
```

followed by a RETURN and each sound channel would be shut off.

I like to make life easy on myself though, so, I simply type in:

```
END
```

followed by a RETURN and that does the trick with the lowest overhead.

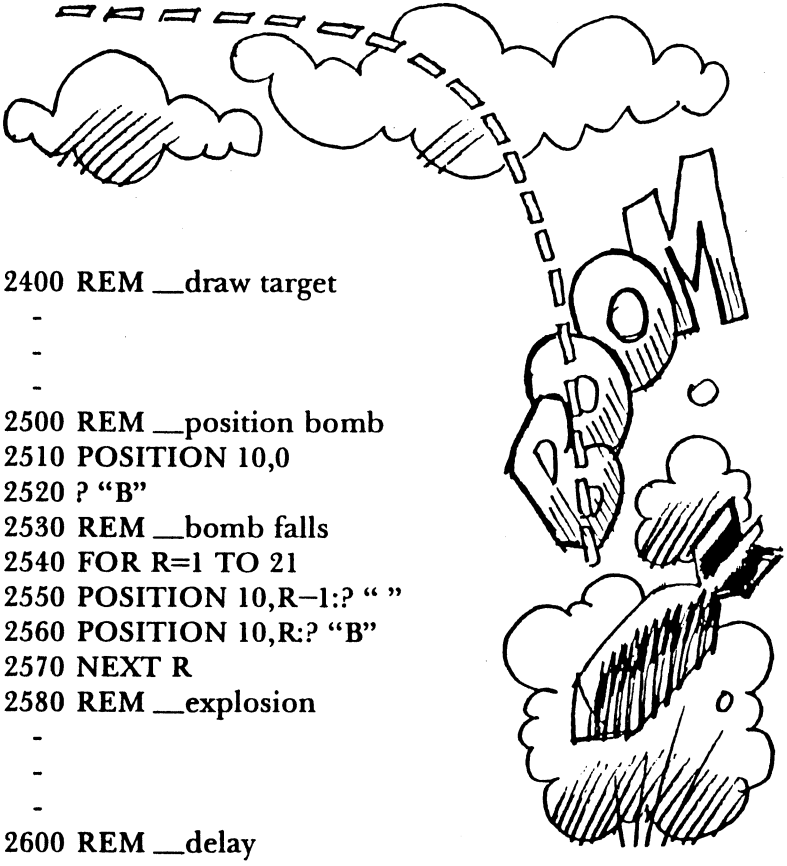
For your own information, there are several other commands that will automatically "turn off" the sound channels. They are:

```
RUN  CLOAD  CSAVE  DOS  NEW
```

However, you can see that executing these commands have consequences of their own.

BORROWING CODE

I would like to encourage you to borrow code from our programs any time you like. For example, each of the menu charts show the line number in the program where the sound effect routine can be found. Let's just review how to isolate one of these sound effects and use it in our own program. As an example, let us say the program we are writing looks something like this:




```
2400 REM __draw target
  .
  .
  .
2500 REM __position bomb
2510 POSITION 10,0
2520 ? "B"
2530 REM __bomb falls
2540 FOR R=1 TO 21
2550 POSITION 10,R-1:? " "
2560 POSITION 10,R:? "B"
2570 NEXT R
2580 REM __explosion
  .
  .
  .
2600 REM __delay
2610 FOR W=1 TO 200:NEXT W
2620 END
```

APPENDIX

As you can see, we would like to insert an explosion sound effect after line 2580. Looking at the MENU1 selections we see there is just such a sound effect and it can be found in the MENU1 program at line number 2300. If we look at the code from line number 2300 in MENU1 it looks like this:

```
2300 REM ****************************
2310 REM *            EXPLOSION     *
2320 REM ****************************
2330 FOR X=0 TO 35

2340 SOUND 0,X,8,14
2350 FOR Y=1 TO 25:NEXT Y:NEXT X
2360 FOR X=0 TO 14

2370 SOUND 0,35+X,8,14-Y
2380 FOR Y=1 TO 25:NEXT Y:NEXT X
2390 RETURN
```

The first thing we must do is simply LIST this routine to an ATASCII file on disk or tape as follows:

For DISK:

```
LIST"D:EXPLODE.LST",2300,2390
```

For TAPE:

```
LIST"C:",2300,2390
```

APPENDIX

Next, we must make sure our BASIC work area is clear of any programming. To do this simply type NEW followed by a RETURN and the work area will be clean. Now, let's get our routine back into memory as follows:

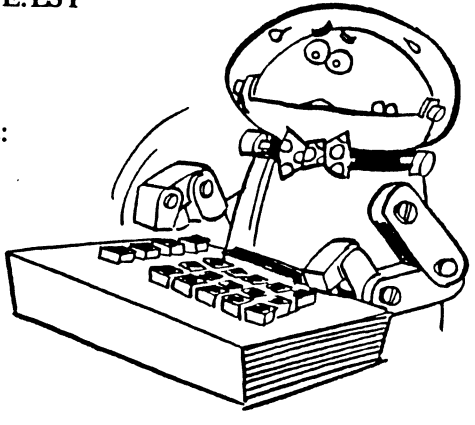
For DISK

ENTER"D:EXPLODE.LST

For TAPE:

position your TAPE:

ENTER"C:



NOTE: when using SAVE, LOAD, LIST, or ENTER, closing quotes (") are not necessary unless line numbers are specified following the command

After successfully ENTERing the code we can LIST it to the screen and we should see:

```
2300 REM ****************************
2310 REM *          EXPLOSION       *
2320 REM ****************************
2330 FOR X=0 TO 35
2340 SOUND 0,X,8,14
2350 FOR Y=1 TO 25:NEXT Y:NEXT X
2360 FOR X=0 TO 14
2370 SOUND 0,35+X,8,14-Y
2380 FOR Y=1 TO 25:NEXT Y:NEXT X
2390 RETURN
```

APPENDIX

The first thing that becomes apparent about this code is that the line numbers are not compatible with our host program. We must fit this routine between lines 2580 and 2600. This is simple when the code will fit on one screen. Clear the screen with a SHIFT CLEAR and list the code with a LIST RETURN. Now wipe it out of memory by typing NEW - RETURN. The code will still be on the screen but will no longer exist in memory. Since we have our own REM statement we are not concerned with the REM statements headlining the borrowed code. Therefore, move the cursor to line number 2330 on the screen and type over it with 2582 followed by a RETURN. Type over old line numbers with each new line number incremented by two (2) until your code looks like this:

```
2300 REM ****************************
2310 REM *          EXPLOSION       *
2320 REM ****************************
2582 FOR X=0 TO 35
2584 SOUND 0,X,8,14
2586 FOR Y=1 TO 25:NEXT Y:NEXT X
2588 FOR X=0 TO 14
2590 SOUND 0,35+X,8,14-Y
2592 FOR Y=1 TO 25:NEXT Y:NEXT X
2390 RETURN
```

Notice we did not renumber line 2390 because our code does not require a RETURN since it is not a sub-routine. Now, if you clear the screen and list the code you should have:

```
2582 FOR X=0 TO 35
2584 SOUND 0,X,8,14
2586 FOR Y=1 TO 25:NEXT Y:NEXT X
2588 FOR X=0 TO 14
2590 SOUND 0,35+X,8,14-Y
2592 FOR Y=1 TO 25:NEXT Y:NEXT X
```

APPENDIX

This code will fit nicely into our host program. The way we do that is to LIST it back onto DISK or TAPE:

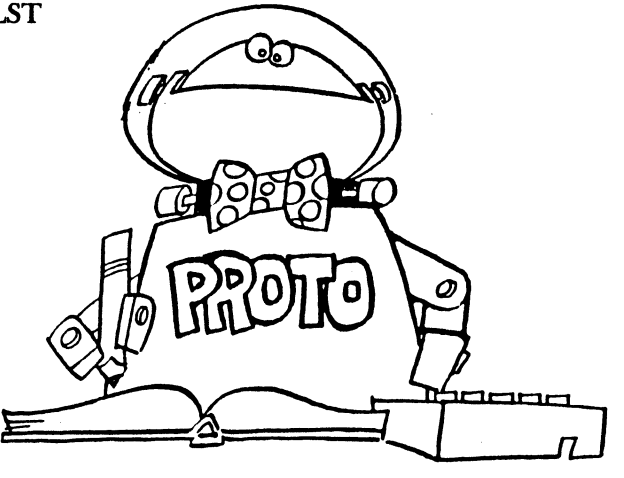
For DISK:

LIST"D:EXPLODE.LST

For TAPE:

position TAPE:

LIST"C



NOTE: line numbers to be listed did not have to be specified as we wanted it all.

Now LOAD the host program and ENTER the sound effect routine

For DISK:

LOAD"D: OURPROG
ENTER "D: EXPLODE.LST

For TAPE:

position TAPE:

LOAD"C:

APPENDIX

When you LIST your final results they should look like this:

```
2400 REM __draw target
   -
   -
   -
2500 REM __position bomb
2510 POSITION 10,0
2520 ? "B"
2530 REM bomb falls
2540 FOR R=1 TO 21
2550 POSITION 10,R-1:? " "
2560 POSITION 10,R:? "B"
2570 NEXT R
2580 REM __explosion
2582 FOR X=0 TO 35
2584 SOUND 0,X,8,14
2586 FOR Y=1 TO 25:NEXT Y:NEXT X
2588 FOR X=0 TO 14
2590 SOUND 0,35+X,8,14-Y
2592 FOR Y=1 TO  25:NEXT Y:NEXT X
2600 REM __delay
2610 FOR W=1 TO 200: NEXT W
2620 END
```

If you wanted to integrate a routine that would not fit on one screen, you would have to renumber it without removing it from memory with the NEW statement and then go back and remove the old code by typing the old line numbers, each followed by a RETURN.

All this can be very exasperating and is easily overcome by obtaining a RENUMBER routine. I highly recommend you get one. We provide a very nice one in our TRICKY TUTORIAL #17, Programming Utilities

CAUTION: WHEN WORKING IN ATARI BASIC ALWAYS CHECK YOUR RESULTS BY LISTING YOUR CHANGES TO THE SCREEN. MANUALLY RENUMBERING CODE CAN SOMETIMES RESULT IN CODE DESTRUCTION. WEIRD, I KNOW!

PROGRAMMING TECHNIQUE

For the purposes of this discussion let's take the following example:

```
1000 FOR PITCH=10 TO 100 STEP 10
1010 FOR DISTORTION=2 TO 14 STEP 2
1020 SOUND 0,PITCH,DISTORTION,10
1030 FOR WAIT=1 TO 200:NEXT WAIT
1040 NEXT DISTORTION
1050 NEXT PITCH
```

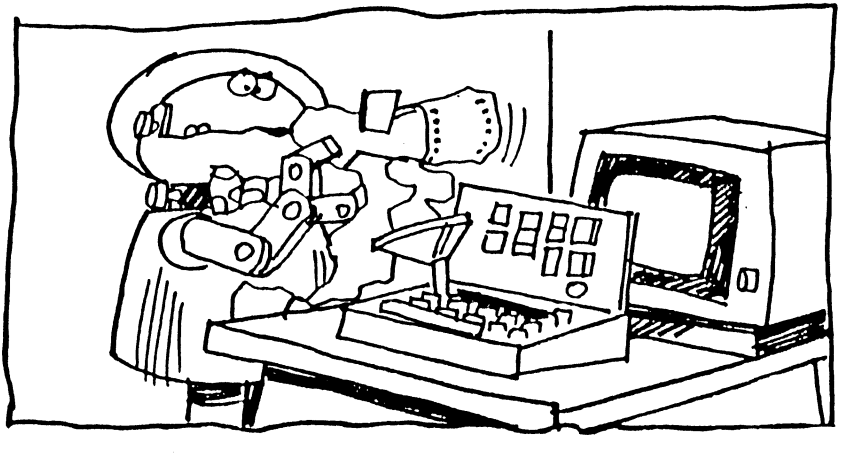
In the above example there are three things that represent good and bad technique. The first is line numbers. There should be little controversy here. Each line number is incremented by 10. That leaves room to insert new code. It also makes it easy to follow and is therefore desirable to have all the line numbers in your final program properly numbered. Unfortunately, ATARI does not provide a RENUMBER routine in their BASIC. I highly recommend you obtain one.

The second point may be somewhat controversial. It concerns the use of variable names. BASIC is not a compiled language and the use of long variable names uses memory. Good programming is logically oriented and should not really require long descriptive variable names, especially if the code is localized, such as our example, and does not refer to other segments of the host program. The example could be re—written like this:

APPENDIX

```
1000 FOR P=10 TO 100 STEP 10
1010 FOR D=2 TO 14 STEP 2
1020 SOUND 0,P,D,10
1030 FOR W=1 TO 200:NEXT W
1040 NEXT D
1050 NEXT P
```

We were able to save 45 BYTES of memory (about 30%) by reducing variable names to single letters. Any competent programmer would have little trouble understanding this code without the aid of explicit variable names. Our savings can be substantial when you are trying desparately to fit your program into a 16K machine.



Finally, I present another controversial point. It concerns multiple statements on a line. I know several programmers that would write our example like this:

```
1000 FOR PITCH=10 TO 100 STEP 10:FOR DISTORTION=2 TO 14 STEP 2:SOUND 0, PITCH, DISTORTION, 10
1010 FOR WAIT=1 TO 200:NEXT WAIT:NEXT DISTORTION:NEXT PITCH
```

If they could have fit it all on one line they would have. The idea here is to save memory and increase execution speed by reducing line numbers. They continue to use long descriptive variable names for the sake of easy maintenance. Good Grief! There are a couple of contradictory arguments. Someone who really believed the first argument reagarding conversation of memory and speed of execution would at least write the example like this:

```
1000 FOR P=10 TO 100 STEP 10:FOR D=2 TO 24 STEP
2:SOUND O,P,D,10:FOR W=1 TO 200:NEXT D:NEXT p
```

Now the added argument that this particular routine is "localized" to one line number and recognizable as a routine "unto itself" is held out to support this kind of technique. I would hate to have to "insert" some additional function into this routine. There is no room to do it. It is far less legible than:

```
1000FOR P=10 TO 100 STEP 10
1010 FOR D=2 TO 14 STEP 2
1020 SOUND 0,P,D,10
1030 FOR W=1 TO 200:NEXT W
1040 NEXT W
1050 NEXT P
```

We lose only 10 BYTES for the additional line numbers (1010 through 1050) and any loss in the speed at which this routine will execute will never be detected by the most discerning user. We gain code that is easy to understand, maintain, and just as easy to isolate. These are only a few obvious touches of good software "engineering" used by professionals when dealing with interpreter languages. See my TRICKY TUTORIAL#18 for a whole range of tips regarding programming techniques, advanced screen and keyboard handling, and coding tricks in BASIC. it won't be long before you're submittting professionally written programs to us for marketing!

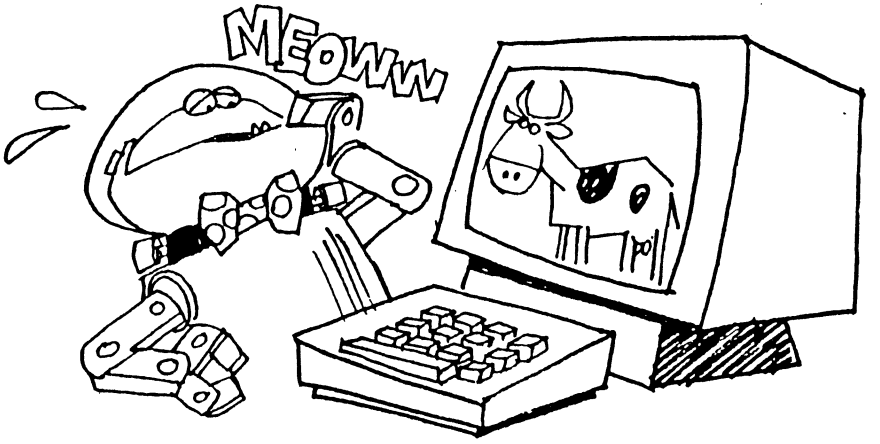
APPENDIX

TIMING SOUND EFFECTS

An interesting aspect of ATARI BASIC is that, depending on where it is situated in the program, a FOR/NEXT loop will run at different speeds. A common technique with inexperienced BASIC programmers is to use a "do nothing" loop to create delays in the execution of their program. Unfortunately, if you are happy with:

```
2030 FOR W=1 TO 1000
2040 NEXT W
```

at the beginning of your program, you will find the delay longer if it is used much later in your code. The reason for this is that each time the loop executes it's internal instructions (in this case there is nothing between lines 2030 and 2040 to do, but it looks for them anyway), it starts at the first line of code in the program and "falls through" to the next line to be executed inside the loop. Therefore if the loop occurs early in the code it does not have as many line numbers to "fall through" as it would if it occurs near the end of the code."For this reason, a sound effect, which usually relies on some kind of loop, will often sound different if placed in two different places in a program.



APPENDIX

Timing is also affected by the display mode being used in your program. For instance, a FOR/NEXT loop may take 30% less time in GRAPHICS mode 3 than GRAPHICS mode O. In fact, it is often desirable to completely turn off the display at the beginning of a program if you have much initialization to perform. This will greatly improve your initialization speed.

There are several ways to avoid these "timing" problems. If the same sound effect is needed more than once in a program it should be called as a ub-routine. Since it will reside only once in the same location inside your program it will always sound the same. If you do this there is still something to watch out for. Programmers often times "build" a program a piece at a time. Early in the development of a program you may create a sound effect sub-routine to be calledthroughout the program. I often place sub-routines at the end of my programs and if I add a lot more code before I am finished then my sound effect may not longer sound right. Once I am done with the program and all the lines of code are in place I will probably want to adjust the timing of my sound effect sub-routine. Of course, you can place that routine at the beginning of your program and any subsquent lines of code will not change its timing.

Another approach is to place a "do nothing" time delay loop at the beginning of your code. For example:

```
1040 FOR W=1 TO WAIT:NEXT W
```

Now I can call this routine with any value I want in the variable WAIT and adjust my time delays accordingly. Both of these techniques are still not entirely reliable. Using the SYSTEM TIMERS built into the computer is a much more desirable approach. These timers are simply memory locations that count backwards every 60th of a second. They are located at the following addresses:

LABEL	DECIMAL	HEX	DESCRIPTION
CDTMV3	540,541	21C,21D	SYSTEM TIMER 3
CDTMV4	542,543	21E,21F	SYSTEM TIMER 4
CDTMV5	544,555	220,221	SYSTEM TIMER 5

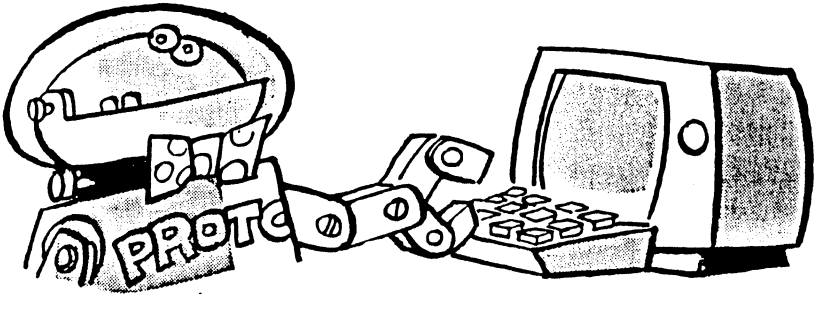
APPENDIX

There are other timers and timer related memory locations but we will only deal with those I listed. Using these timers requires little thought. Each timer is made up of two memory locations. Each location is capable of holding a value from 0 to 255. The first memory location is called the LOW ORDER BYTE and is counted down before the next, or HIGH ORDER BYTE, is counted down. If I place a value of 255 in the LOW ORDER LOCATION 540 it will take about 4.25 seconds to decrement (cound down) to zero. If it counts down once each 60th of a second then 255/60=4.25. If your timing does not require anything over 4.25 seconds then the following code would work fine:

```
1080 POKE 540,120:REM 60*2=120 seconds
1090 IF PEEK(540)0 THEN 1120
1100 GOTO 1090
1110 REM continue with program
1120 ? "hello"
```

This could just have easily been put into a sub–routine

```
1080 POKE 540,120:REM 60*2=120 seconds
1090 GOSUB 4000
.
.
.
.
4000 IF PEEK(540)=0 THEN RETURN
4010 GOTO 4000
```

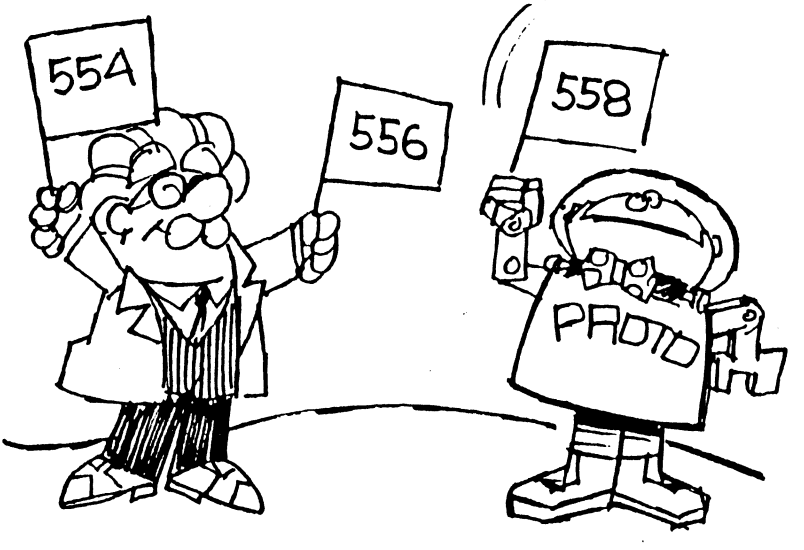


APPENDIX

It is important that location 541 contains a zero (0) value. It normally does unless you have inadvertently put a value into it and it could not count down before reaching line 1080. Any value in the HIGH ORDER location of a SYSTEM TIMER causes something unique to happen. For instance, if any value is in location 541 then as soon as 540 reaches zero (0) 541 will count down one and 540 will be set to 255 and start counting down again. The problem is, depending on your code, you may not PEEK at 540 exactly when it is a zero (0) value.

Longer delays present a slightly different programming challenge. Just as there are three timers there are three memory locations that hold a "FLAG" for each timer. They are:

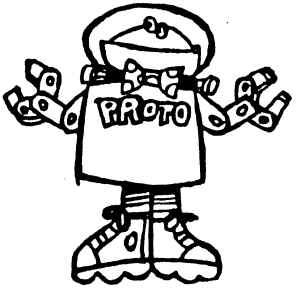
LABEL	DECIMAL	HEX	DESCRIPTION
CDTMF3	554	22A	SYSTEM TIMER 3 FLAG
CDTMF4	556	22C	SYSTEM TIMER 4 FLAG
CDTMF5	558	22E	SYSTEM TIMER 5 FLAG



APPENDIX

If you place a positive value, say a 1, in location 554 it will be set to zero(0) when the SYSTEM TIMER 3 locations have counted to zero. This is commonly called a "FLAG" because the value will change to specific value depending on the operation of some memory location. In this case the location is the SYSTEM TIMER. It is wiser to test this FLAG than to test location 540. The reason is that you do not have to test both the HIGH ORDER BYTE and the LOW ORDER BYTE for zero to terminate the time delay. Here is a timer that works in seconds only

```
1030 REM __ ask how many seconds
1040 ? "How many seconds do you want";
1050 INPUT SECONDS


1060 REM __ a JIFFY is 1/60th of a second
1070 REM __ the counters work in jiffies
1080 JIFFIES=SECONDS*60


1090 REM __ get HI and LO values
1100 HI=INT(JIFFIES/255)
1110 LO=JIFFIES-(HI*255)

1120 REM __ set system timers
1130 POKE 554,1
1140 POKE 540,LOW:POKE 541,HIGH


1150 REM __ start timing demo
1160 ? "STARTING NOW!"
1170 IF PEEK(554)=0 THEN 1190
1180 GOTO 1170
1190 ? "DONE":END
```



Terrific, you say, um, but what are the drawbacks. Well, if you are able to work in 60ths of a second you're all set, otherwise, your only alternatives are to write an assembly language routine that works in smaller increments and do a USR call to it. Also, system timers do not lend themselves easily to the use of a BASIC STEP statement. You could imbed a system timer into a FOR/NEXT loop and change the variable which you POKE into it, but it will still be affected by where in the code the routine is placed. You're better off using a FOR/NEXT loop with a STEP statement and "fine tune" your sound effects when your entire program is done by adjusting the time delays. Oh well.

SOUND EFFECTS PROGRAM LISTINGS

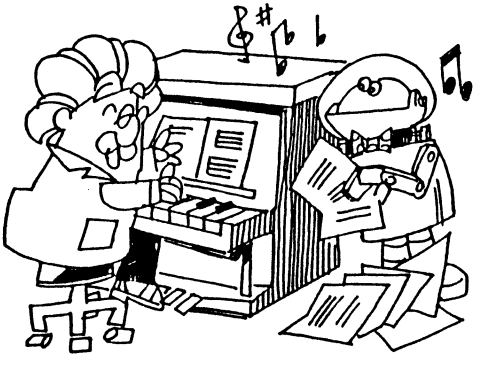
This section contains selected program listings of the SOUND EFFECTS software for you to study. The assembly code has been stripped leaving only the BASIC code. If you desire to see the assembly code of the program currently in memory, press the BREAK key, type LIST, and press RETURN. The code should scroll down your screen. To stop and start the scrolling, press the CTRL key and number 1 simultaneously.

TRAIN

```
1000 REM
2000 REM
2010 DIM WT$(57)
    :REM __TDELAY.OBJ

2050 REM
2060 DIM FM$(54)
    :REM __FILMEM.OBJ

2100 REM
2110 DIM MM$(49)
    :REM __MOVMEM.OBJ

2150 REM
2160 DIM SS$(95)
    :REM __SOUNDS.OBJ

2220 REM
2230 DIM CC$(37)
    :REM __CIOCTL.OBJ

2260 REM
2270 DIM SP$(48)
    :REM __SETPMS.OBJ

2310 REM
2320 DIM MP$(118)
    :REM __MOVPMS.OBJ

2390 REM
2400 DIM BS$(96)
    :REM __PROTO

2460 REM
2470 DIM LK$(19)
    :REM __LOCKIT.OBJ

2490 REM
4000 REM [redacted]
4010 REM
4020 REM __memory mgt & screen
4030 DIM M$(80),T$(40)
4040 T$="#"
    :T$(40)="#"
    :T$(2)=T$
4050 P106=PEEK(106)
4060 PB=P106-4
    :PA=PB*256
4070 POKE 106,PB
```

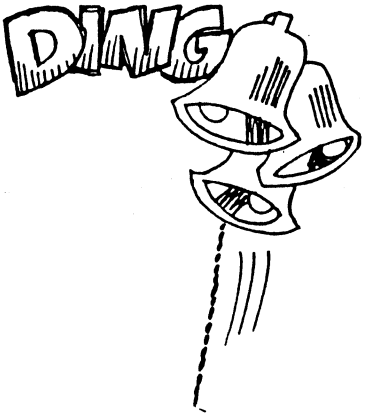
```
4080 GRAPHICS 0
     :POKE 752,1
4090 SETCOLOR 2,0,8
     :SETCOLOR 1,0,0
4100 U=USR(ADR(FM$),PA,0,1023)
4110 U=USR(ADR(MM$),ADR(BS$),PA,96)
4120 U=USR(ADR(SP$),PB)
4130 REM __paint title screen
4140 POKE 82,1
     :POSITION 1,0
4150 ? "┌──────────────────────────────────────┐"
4160 ? "|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|"
4170 ? "|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|"
4180 ? "|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|"
4190 ? "|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|"
4200 ? "|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|"
4210 ? "|~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|"
4220 ? "|▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀|"
4230 ? "|▀~~~▀~▀~▀~▀~▀~▀~~~~~~▀~~~▀~▀~▀~▀~▀~▀|"
4240 ? "|▀~~~▀▀▀▀▀~▀~▀~▀~~~~~~▀~~~▀▀▀▀▀~▀~▀|"
4250 ? "|▀~~~▀~▀~▀~▀~▀~▀~~~~~~▀~~~▀~▀~▀~▀~▀~▀|"
4260 ? "|▀▀▀▀▀~▀~▀▀▀▀~▀▀▀▀~~~~~~▀▀▀▀~▀~▀▀▀▀~▀▀▀|"
4270 ? "└──────────────────────────────────────┘"
4280 REM __bring out PROTO
4290 HF=20
     :HT=100
     :BF=0
     :BT=2
4300 GOSUB 7330
4310 FOR I=1 TO 4
     :GOSUB 7210
     :NEXT I
4320 HF=H
     :HT=190
     :BF=0
     :BT=2
4330 GOSUB 7330
4340 U=USR(ADR(WT$),50)
     :GOSUB 7210
4350 POSITION 0,20
     :? T$
4360 U=USR(ADR(SS$),1)
4370 U=USR(ADR(WT$),100)
     :GOSUB 7210
4380 POKE 82,2
     :POSITION 2,14
4390 ? "~ ╓ ~+~~~~~ ▲ ~ ╓ ~ ▲ ~~~~=~~=~~~~~~~●~~~"
4400 ? "┌┴┴┴┐~~~▄▄▄▄H/──\~~~=~~=~~~●~~|~●~|"
4410 ? "|~~~~├──┤~|╓|H|~~|~~~|~~|~~~|~●~●|~●"
4420 ? "|~~~~|~~|||||||||~~|~~~|~~|~~~~~|~|~~|"
4430 U=USR(ADR(SS$),1)
4440 U=USR(ADR(WT$),100)
     :GOSUB 7210
4450 U=USR(ADR(WT$),100)
4460 HF=H
     :HT=230
     :BF=0
     :BT=2
4470 GOSUB 7330
4480 GOSUB 7210
     :U=USR(ADR(WT$),140)
4490 GOSUB 7210
     :U=USR(ADR(WT$),140)
4500 REM __start train ride
4510 W=70
     :GOSUB 7020
```

```
4520 GOSUB 7090
4530 FOR W=37 TO 1 STEP -1
4540 GOSUB 7020
     :GOSUB 7020
     :W=W-M
4550 NEXT W
4560 REM __runnin on the track
4570 FOR H=220 TO 28 STEP -3
4580 IF H<>121 THEN 4650
4590 GOSUB 7090
     :SOUND 0,0,0,0
4600 U=USR(ADR(MP$),0,1,89,H-B)
4610 U=USR(ADR(SS$),1)
4620 U=USR(ADR(WT$),100)
4630 FOR I=1 TO 3
     :GOSUB 7210
     :NEXT I
4640 U=USR(ADR(WT$),100)
4650 FOR B=9 TO 11
4660 U=USR(ADR(MP$),0,B,89,H-B)
4670 GOSUB 7020
4680 NEXT B
4690 NEXT H
4700 REM __end train ride
4710 FOR W=0 TO 8
     :GOSUB 7020
     :NEXT W
4720 FOR W=10 TO 16
4730 GOSUB 7020
     :GOSUB 7020
     :W=W+M
4740 NEXT W
4750 W=44
     :GOSUB 7020
4760 SOUND 0,0,0,0
4770 GOSUB 7210
     :U=USR(ADR(WT$),200)
4780 POKE 106,P106
     :GRAPHICS 0
4790 RUN "D:EFFECTS"
4800 END
7000 REM  SUB-ROUTINES
7010 REM __choo choo sound
7020 M=INT(W/10)
7030 FOR J=8 TO 2 STEP -1
7040 SOUND 0,0,0,J
7050 U=USR(ADR(WT$),W)
7060 NEXT J
7070 RETURN
7080 REM __whistle
7090 FOR I=100 TO 200 STEP 100
7100 SOUND 1,39,14,8
7110 SOUND 2,50,14,12
7120 SOUND 3,33,14,4
7130 U=USR(ADR(WT$),I)
7140 FOR J=1 TO 3
7150 SOUND J,0,0,0
7160 NEXT J
7170 U=USR(ADR(WT$),20)
7180 NEXT I
7190 RETURN
7200 REM __voice of PROTO
7210 READ M$
7220 L=LEN(M$)
     :TD=L*10
     :CM=(40-L)/2-1
7230 POSITION CM,22
7240 FOR V=1 TO L
7250 SOUND 0,ASC(M$(V,V)),10,10
7260 ? CHR$(ASC(M$(V,V)));
7270 NEXT V
7280 SOUND 0,0,0,0
7290 U=USR(ADR(WT$),INT(TD*2.5))
7300 POSITION CM,23
7310 ? "↑■"
     :RETURN
7320 REM __move PROTO
7330 FOR H=HF TO HT STEP 3
7340 FOR B=BF TO BT
7350 U=USR(ADR(MP$),0,B,89,H+0)
7360 SOUND 0,B*20,10,10
7370 U=USR(ADR(WT$),4)
7380 NEXT B
7390 NEXT H
7400 U=USR(ADR(MP$),0,1,89,H)
7410 SOUND 0,0,0,0
7420 RETURN
7430 REM
8000 REM  DATA
8010 DATA Psst.. wanna ride my train?
8020 DATA Sure you do.  Sure you do!
8030 DATA Got a good imagination?
8040 DATA Good!  Go get your luggage.
8050 DATA Gimme some tracks!
8060 DATA And a little scenery too!
8070 DATA Be right back....
8080 DATA Gotta get the TRAIN!
8090 DATA READY?
8100 DATA Hey.... gimme a break!
8110 DATA It's only make believe....
8120 DATA Good Grief!
8130 DATA NO!  No ticket refunds!!
```



MENU1

```
0 REM
  SOUNDSUB (C) 1982 BY JERRY WHITE
5 OFF=6
    :WAIT=7
    :GOTO 4000
6 FOR SHUTUP=0 TO 3
    :SOUND SHUTUP,0,0,0
    :NEXT SHUTUP
    :RETURN
7 POKE 540,JIFFY
8 IF PEEK(540) THEN 8
9 RETURN
1100 REM ***************************
1110 REM *       YOU WIN SOUND       *
1120 REM ***************************
1130 FOR X=150 TO 45 STEP -1
    :SOUND 0,X,10,15
    :NEXT X
1140 FOR X=1 TO 100
    :SOUND 0,45,10,15
1150 NEXT X
1160 RETURN
1200 REM ***************************
1210 REM *       YOU LOSE SOUND      *
1220 REM ***************************
1230 FOR X=150 TO 255
   :SOUND 0,X,10,15
    :NEXT X
    :FOR X=1 TO 100
1240 SOUND 0,255,10,15
    :NEXT X
1250 RETURN
1300 REM ***************************
1310 REM *         ????              *
1320 REM ***************************
1330 FOR I=1 TO 5
    :FOR J=10 TO 5 STEP -1
    :SOUND 0,J,14,8
    :SOUND 1,J,2,8
    :NEXT J
    :SOUND 0,0,0,0
1340 FOR J=1 TO RND(0)*30
    :NEXT J
    :NEXT I
    :RETURN
1400 REM ***************************
1410 REM *        YOU'RE A WINNER    *
1420 REM ***************************
1430 FOR X=1 TO 10
1440 FOR Y=40 TO 90 STEP 4
1450 SOUND 0,Y,10,5
    :SOUND 1,100-Y,10,12
1460 NEXT Y
    :NEXT X
1470 SOUND 0,0,0,0
    :SOUND 1,0,0,0
1480 SOUND 0,0,0,0
    :SOUND 1,0,0,0
1490 RETURN
1500 REM ***************************
1510 REM *         HORN1             *
1520 REM ***************************
1530 FOR Y=1 TO 5
1540 SOUND 0,30,12,12
    :SOUND 1,31,12,12
1550 FOR X=1 TO 40
    :NEXT X
1560 SOUND 0,0,0,0
    :SOUND 1,0,0,0
1570 FOR X=1 TO 5
    :NEXT X
1580 NEXT Y
1590 RETURN
1600 REM ***************************
1610 REM *          WHAM SOUND       *
1620 REM ***************************
1630 FOR A=15 TO 1 STEP -1
1640 SOUND 1,10,8,A
1650 NEXT A
1660 RETURN
1700 REM ***************************
1710 REM *          DOOR SLAMMING    *
1720 REM ***************************
1730 FOR A=15 TO 1 STEP -1
1740 SOUND 1,20,6,A
    :SOUND 0,10,8,A
1750 NEXT A
1760 RETURN
1800 REM ***************************
1810 REM *         ELECTRIC BELL     *
1820 REM ***************************
1830 FOR I=1 TO 30
1840 FOR U=1 TO 4
    :SOUND 0,20,10,8
1850 FOR WAIT=1 TO 5
    :NEXT WAIT
1860 SOUND 0,0,0,0
1870 NEXT I
1880 RETURN
1900 REM ***************************
1910 REM *       SLIDE DOWN CHUTE    *
1920 REM ***************************
1930 FOR P=0 TO 250 STEP 5
1940 SOUND 1,P/5,8,8
    :SOUND 0,P,10,4
1950 SOUND 1,P/5,8,4
    :NEXT P
1960 FOR V=15 TO 0 STEP -1
1970 SOUND 0,102,12,V
    :SOUND 1,102,12,V
1980 NEXT V
1990 RETURN
2000 REM ***************************
2010 REM *              FALLING      *
2020 REM ***************************
2030 FOR X=0 TO 150
2040 SOUND 0,X,10,10
2050 FOR Y=1 TO 5
    :NEXT Y
    :NEXT X
2060 FOR X=15 TO 0 STEP -1
2070 SOUND 0,10,8,X
2080 FOR Y=1 TO 30
    :NEXT Y
2090 RETURN
```




```
2100 REM ***************************
2110 REM *         HAPPY           *
2120 REM ***************************
2130 FOR Y=1 TO 6
2140 FOR X=7 TO 50 STEP 2
2150 SOUND 0,X,10,10
2160 NEXT X
2170 FOR X=50 TO 7 STEP -2
2180 SOUND 0,X,10,10
2190 NEXT X
    :NEXT Y
    :RETURN
2200 REM ***************************
2210 REM *  RANDOM PING-PONG NOISE *
2220 REM ***************************
2230 U=INT(RND(0)*100)+50
    :SOUND 0,U,10,8
2240 FOR W=1 TO 5
    :NEXT W
2250 SOUND 0,0,0,0
2260 RETURN
2300 REM ***************************
2310 REM *          EXPLOSION      *
2320 REM ***************************
2330 FOR X=0 TO 35
2340 SOUND 0,X,8,14
2350 FOR Y=1 TO 25
    :NEXT Y
    :NEXT X
2360 FOR X=0 TO 14
2370 SOUND 0,35+X,8,14-X
2380 FOR Y=1 TO 25
    :NEXT Y
    :NEXT X
2390 RETURN
2400 REM ***************************
2410 REM *       BEEP BEEP NOISE   *
2420 REM ***************************
2430 FOR I=1 TO 2
2440 SOUND 0,12,12,8
    :FOR W=1 TO 20
2450 NEXT W
    :SOUND 0,0,0,0
2460 NEXT I
2470 RETURN
2500 REM ***************************
2510 REM *        DEE DUMP         *
2520 REM ***************************
2530 SOUND 0,65,12,10
    :FOR Z=1 TO 60
2540 NEXT Z
    :SOUND 0,126,12,10
2550 FOR Z=1 TO 100
    :NEXT Z
2560 SOUND 0,0,0,0
2570 RETURN
2600 REM ***************************
2610 REM *           HORN2         *
2620 REM ***************************
2630 D=4
2640 SOUND 0,121,10,8
2650 SOUND 1,128,10,8
2660 SOUND 2,8,2,2
2670 FOR J=1 TO D*70
    :NEXT J
2680 RETURN
```

```
2700 REM ***************************
2710 REM *         BUZZER          *
2720 REM ***************************
2730 D=1
2740 SOUND 0,42,2,15
2750 FOR J=1 TO D*200
    :NEXT J
2760 RETURN
2800 REM ***************************
3000 POKE 764,255
    :GRAPHICS 0
    :RUN "D:MENU2"
4000 DIM C$(13)
    :C$="your~choice~?"
5000 GRAPHICS 17
    :SETCOLOR 4,10,0
    :SETCOLOR 2,9,12
    :SETCOLOR 0,1,12
    :POKE 559,0
    :SOUND 0,0,0,0
5020 POSITION 0,0
    :? #6;"~~effects~menu~#~1~"
    :? #6
5100 ? #6;"~~A.~[illegible]"
5200 ? #6;"~~B.~[illegible]"
5300 ? #6;"~~C.~[illegible]"
5400 ? #6;"~~D.~[illegible]"
5500 ? #6;"~~E.~[illegible]"
5600 ? #6;"~~F.~[illegible]"
5700 ? #6;"~~G.~[illegible]"
5800 ? #6;"~~H.~[illegible]"
5900 ? #6;"~~I.~[illegible]"
6000 ? #6;"~~J.~[illegible]"
6100 ? #6;"~~K.~[illegible]"
6200 ? #6;"~~L.~[illegible]"
6300 ? #6;"~~M.~[illegible]"
6400 ? #6;"~~N.~[illegible]"
6500 ? #6;"~~O.~[illegible]"
6600 ? #6;"~~P.~[illegible]"
6700 ? #6;"~~Q.~[illegible]"
7000 ? #6;"~~R.~[illegible]"
    :POKE 559,34
7100 GOSUB 9000
    :POKE 764,255
7200 POKE 764,255
    :OPEN #1,4,0,"K:"
7300 GET #1,KEY
    :CLOSE #1
7400 IF KEY<65 OR KEY>82 THEN 8000
7500 POSITION 17,23
    :? #6;CHR$(KEY);
7600 POSITION 2,KEY-63
    :? #6;CHR$(KEY+128);
    :JIFFY=15
    :GOSUB WAIT
7700 GOSUB (KEY-64)*100+1000
    :GOSUB OFF
7800 POSITION 2,KEY-63
    :? #6;CHR$(KEY);
7900 POSITION 17,23
    :? #6;"?";
    :GOTO 7100
8000 POSITION 17,23
    :? #6;"~";
8100 FOR JW=15 TO 0 STEP -0.5
    :SOUND 0,102,12,JW
    :SOUND 1,51,12,JW
    :POKE 709,JW
    :NEXT JW
8200 POKE 709,202
    :GOTO 7100
9000 JIFFY=0
    :FOR X=17 TO 5 STEP -1
    :POSITION X,23
    :? #6;"[illegible]";
    :GOSUB WAIT
    :POSITION X,23
    :? #6;"[illegible]";
    :NEXT X
9050 JIFFY=6
    :GOSUB WAIT
9100 JIFFY=0
    :FOR X=1 TO 13
    :POSITION X+4,23
    :? #6;"[illegible]";
    :GOSUB WAIT
    :POSITION X+4,23
    :? #6;C$(X,X);
    :NEXT X
    :RETURN
```

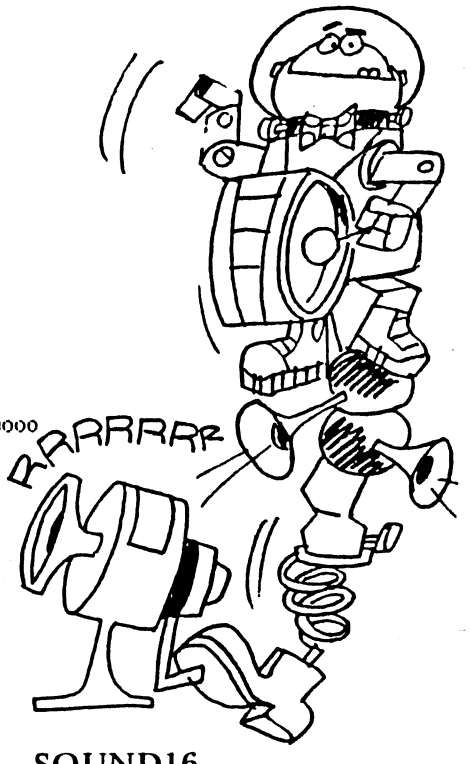
MENU2

```
0 REM
  SOUNDSUB (c) 1982 by Jerry White
5 OFF=6
    :WAIT=7
    :GOTO 4000
6 FOR SHUTUP=0 TO 3
    :SOUND SHUTUP,0,0,0
    :NEXT SHUTUP
    :RETURN
7 POKE 540,JIFFY
8 IF PEEK(540) THEN 8
9 RETURN
100 REM [illegible]
110 FOR JW=100 TO 0 STEP -5
120 SOUND 0,JW,4,JW/10
130 SOUND 0,0,0,0
140 NEXT JW
190 RETURN
200 REM [illegible]
220 X=0.1
    :FOR JW=1 TO 100
    :FOR LOUD=15 TO 5 STEP -X
    :SOUND 0,15,0,LOUD
    :NEXT LOUD
230 X=X+0.01
    :IF X>0.7 THEN X=0.7
240 NEXT JW
```

```
    :SOUND 0,0,0,0
260 FOR ME=1 TO 2
    :SOUND 1,40,10,10
    :SOUND 2,10,10,8
    :SOUND 3,90,10,10
270 JIFFY=30
    :GOSUB WAIT
    :GOSUB OFF
280 JIFFY=15
    :GOSUB WAIT
    :NEXT ME
290 RETURN
300 REM [illegible]
310 FOR TWINKLE=1 TO 20
    :P=(RND(0)*10)+2
    :POKE 53760,P
    :POKE 53762,P+2
320 FOR V=166 TO 160 STEP -3
    :POKE 53761,V
    :POKE 53763,V
    :NEXT V
    :NEXT TWINKLE
390 RETURN
400 REM [illegible]
410 FOR P=0 TO 250 STEP 5
    :SOUND 1,P/5,8,8
    :SOUND 0,P,10,4
    :SOUND 1,P/5,8,4
    :NEXT P
420 FOR V=15 TO 0 STEP -1
    :SOUND 0,102,12,V
    :SOUND 1,102,12,V
    :NEXT V
490 RETURN
500 REM [illegible]
510 FOR SQUEEK=1 TO 3
    :FOR P=15 TO 11 STEP -4
    :FOR V=1 TO 13 STEP 4
    :SOUND 0,P,10,V
    :POKE 53768,13
    :NEXT V
520 FOR V=15 TO 0 STEP -4
    :SOUND 0,P,12,V
    :POKE 53768,13
    :NEXT V
    :NEXT P
530 GOSUB OFF
    :NEXT SQUEEK
590 RETURN
600 REM [illegible]
610 FOR X=30 TO 250 STEP 30
    :POKE 53768,224
    :POKE 53761,98
    :POKE 53760,X
    :SOUND 1,0,8,1
620 POKE 53761,96
    :JIFFY=2
    :GOSUB WAIT
640 NEXT X
650 SOUND 0,1,8,15
    :SOUND 1,255,0,15
    :JIFFY=2
    :GOSUB WAIT
670 FOR V=15 TO 0 STEP -3
    :SOUND 0,25,4,V
    :SOUND 1,25,4,V
    :NEXT V
690 RETURN
700 REM [illegible]
710 FOR P=90 TO 45 STEP -15
    :FOR JW=1 TO 100
720 FOR V=3 TO 13 STEP 5
    :SOUND 0,P,2,V
    :SOUND 1,V,4,2
    :NEXT V
730 NEXT JW
    :SOUND 1,0,0,0
    :NEXT P
790 RETURN
800 REM [illegible]
810 SOUND 3,24,4,12
    :SOUND 0,14,12,12
    :SOUND 1,29,10,12
    :SOUND 2,2,2,12
    :SOUND 3,0,0,0
820 JIFFY=3
    :GOSUB WAIT
    :GOSUB OFF
    :GOSUB WAIT
840 FOR JW=8 TO 0 STEP -1
    :SOUND 3,24-JW,4,JW
    :SOUND 0,10-JW,12,JW
    :SOUND 1,46-JW,10,JW
850 JIFFY=1
    :GOSUB WAIT
    :GOSUB OFF
    :NEXT JW
890 RETURN
900 REM [illegible]
910 POKE 559,0
    :FOR JW=0 TO 50
    :POKE 53279,0
    :POKE 53279,8
    :NEXT JW
    :POKE 559,34
920 FOR JW=0 TO 50
930 POKE 53279,0
    :POKE 53279,8
940 NEXT JW
990 RETURN
1000 REM [illegible]
1010 GRAPHICS 0
    :POKE 559,0
1020 POKE 66,1
    :? CHR$(253);
    :POKE 66,0
1090 RUN
1100 REM [illegible]
1105 FOR D=15 TO 0 STEP -0.5
1110 SOUND 0,150-D*10,0,D
    :SOUND 1,150-D,8,15
    :SOUND 0,0,0,0
1115 NEXT D
1120 FOR D=10 TO 1 STEP -0.1
    :SOUND 1,D,8,D*2
    :SOUND 2,D*2+1,8,D
    :SOUND 3,D*2+2,8,D
    :NEXT D
1125 SOUND 1,0,8,2
    :SOUND 3,0,0,0
    :POKE 540,30
1130 IF PEEK(540) THEN 1130
1135 SOUND 2,0,0,0
```

```
    :POKE 540,30
1140 IF PEEK(540) THEN 1140
1145 SOUND 1,0,8,1
    :POKE 540,30
1150 IF PEEK(540) THEN 1150
1155 POKE 540,120
    :FOR ME=53760 TO 53768
    :POKE ME,0
    :NEXT ME
1160 IF PEEK(540) THEN 1160
1190 RETURN
1200 REM [illegible]
1210 FOR BEAT=1 TO 10
1220 SOUND 0,0,1,15
    :JIFFY=20
    :GOSUB WAIT
1230 SOUND 0,0,0,0
    :JIFFY=40
    :GOSUB WAIT
1240 NEXT BEAT
1290 RETURN
1300 REM [illegible]
1310 FOR JW=1 TO 5
1320 FOR P=5 TO 100 STEP INT(RND(0)*5)+1
1330 SOUND 0,P,8,(RND(0)*10+5)/(0.1*P)
1340 SOUND 1,P+20,8,(RND(0)*10+5)/(0.1*P)
1350 NEXT P
    :NEXT JW
    :GOSUB OFF
1390 RETURN
1400 REM [illegible]
1410 FOR JW=1 TO 2
    :FOR CHIRP=1 TO 2
1420 FOR P=15 TO 0 STEP -1
    :SOUND 0,P,10,P
    :NEXT P
1430 NEXT CHIRP
    :JIFFY=30
    :GOSUB WAIT
    :NEXT JW
1490 RETURN
1500 REM [illegible]
1510 JIFFY=120
    :FOR JW=1 TO 5
1520 FOR P=77 TO 0 STEP -7
    :SOUND 0,P,10,P
    :NEXT P
1530 SOUND 0,0,0,0
    :GOSUB WAIT
    :NEXT JW
1590 RETURN
1600 REM [illegible]
1610 FOR JW=1 TO 26
1620 SOUND 0,26-JW,8,JW/4
    :SOUND 0,0,0,0
1630 JIFFY=1
    :GOSUB WAIT
    :NEXT JW
1690 RETURN
1700 REM [illegible]
1710 FOR JW=0 TO 1
1720 FOR P=0 TO 25 STEP 0.5
1730 SOUND 0,P,8,JW+1
    :POKE 53768,80
    :POKE 53768,0
1740 JIFFY=RND(0)*10
    :GOSUB WAIT
1750 NEXT P
1760 FOR P=26 TO 0 STEP -0.5
1770 SOUND 0,P,8,JW+1
    :POKE 53768,80
    :POKE 53768,0
1780 JIFFY=RND(0)*10
    :GOSUB WAIT
1790 NEXT P
    :NEXT JW
    :GOSUB OFF
    :RETURN
1800 REM [illegible]
1810 FOR R=1 TO 2
    :FOR ME=1 TO 30
    :SOUND 0,25,10,1
    :SOUND 1,0,2,1
    :FOR J=1 TO 2
    :SOUND 0,25,10,8
    :SOUND 1,0,2,8
1820 NEXT J
    :GOSUB OFF
    :NEXT ME
1830 FOR V=7 TO 0 STEP -0.1
    :SOUND 0,25,10,V
    :SOUND 1,0,2,V
    :NEXT V
1840 JIFFY=60
    :GOSUB WAIT
    :NEXT R
1890 RETURN
1900 REM [illegible]
1910 FOR RAIN=0 TO 15 STEP 0.02
1920 SOUND 1,0,0,RAIN
1930 NEXT RAIN
1940 FOR RAIN=0 TO 15 STEP 0.02
1950 SOUND 0,0,0,RAIN
1960 NEXT RAIN
1970 FOR RAIN=15 TO 0 STEP -0.05
1980 SOUND 0,0,0,RAIN
    :SOUND 1,0,0,RAIN
1990 NEXT RAIN
    :RETURN
2000 POKE 764,255
    :GRAPHICS 0
    :RUN "D:FORNEXT"
4000 DIM C$(13)
    :C$="your^choice^?"
5000 GRAPHICS 17
    :SETCOLOR 4,10,0
    :SETCOLOR 2,9,12
    :SETCOLOR 0,1,12
    :POKE 559,0
    :SOUND 0,0,0,0
5020 POSITION 0,0
    :? #6;"^^effects^menu^#^2^^"
    :? #6
5100 ? #6;"^^A.^[illegible]"
5200 ? #6;"^^B.^[illegible]"
5300 ? #6;"^^C.^[illegible]"
5400 ? #6;"^^D.^[illegible]"
5500 ? #6;"^^E.^[illegible]"
5600 ? #6;"^^F.^[illegible]"
5700 ? #6;"^^G.^[illegible]"
```

```
5800 ? #6;"^^H.^[illegible]"
5900 ? #6;"^^I.^[illegible]"
6000 ? #6;"^^J.^[illegible]"
6100 ? #6;"^^K.^[illegible]"
6200 ? #6;"^^L.^[illegible]"
6300 ? #6;"^^M.^[illegible]"
6400 ? #6;"^^N.^[illegible]"
6500 ? #6;"^^O.^[illegible]"
6600 ? #6;"^^P.^[illegible]"
6700 ? #6;"^^Q.^[illegible]"
6800 ? #6;"^^R.^[illegible]"
6900 ? #6;"^^S.^[illegible]"
7000 ? #6;"^^T.^[illegible]"
    :POKE 559,34
7100 GOSUB 9000
    :POKE 764,255
7200 POKE 764,255
    :OPEN #1,4,0,"K:"
7300 GET #1,KEY
    :CLOSE #1
7400 IF KEY<65 OR KEY>84 THEN 8000
7500 POSITION 17,23
    :? #6;CHR$(KEY);
7600 POSITION 2,KEY-63
    :? #6;CHR$(KEY+128);
    :JIFFY=15
    :GOSUB WAIT
7700 GOSUB (KEY-64)*100
    :GOSUB OFF
7800 POSITION 2,KEY-63
    :? #6;CHR$(KEY);
7900 POSITION 17,23
    :? #6;"?";
    :GOTO 7100
8000 POSITION 17,23
    :? #6;"^";
8100 FOR JW=15 TO 0 STEP -0.5
    :SOUND 0,102,12,JW
    :SOUND 1,51,12,JW
    :POKE 709,JW
    :NEXT JW
8200 POKE 709,202
    :GOTO 7100
9000 JIFFY=0
    :FOR X=17 TO 5 STEP -1
    :POSITION X,23
    :? #6;"[illegible]";
    :GOSUB WAIT
    :POSITION X,23
    :? #6;"[illegible]";
    :NEXT X
9050 JIFFY=6
    :GOSUB WAIT
9100 JIFFY=0
    :FOR X=1 TO 13
    :POSITION X+4,23
    :? #6;"[illegible]";
    :GOSUB WAIT
    :POSITION X+4,23
    :? #6;C$(X,X);
    :NEXT X
    :RETURN
```



SOUND16

```
1000 REM ************************
1010 REM *                      *
1020 REM *      SOUND16.JW      *
1030 REM *   by Jerry White 83  *
1040 REM *                      *
1050 REM ************************
1060 REM [illegible]
1070 GRAPHICS 18
    :? #6
    :? #6;"~~~~16~BIT~SOUND"
    :? #6
1080 DIM S16$(56)
    :REM [illegible]
1085 DIM F$(36)
1090 FOR ME=1 TO 56
    :READ BYTE
    :S16$(ME,ME)=CHR$(BYTE)
    :NEXT ME
1100 REM [illegible]
1110 SOUND 0,0,0,0
    :POKE 53768,120
1120 DIM N$(24),FREQ(7,12)
    :N$="B~A#A~G#G~F#F~E~D#D~C#C~"
    :GOTO 1180
1130 REM [illegible]
```

```
1140 POSITION 6,5
     :? #6;"OCTAVE=";OCTAVE;
     :IF BOTH THEN ? #6;"+";OCTAVE+1;
1150 POSITION 6,7
     :? #6;"PITCH=";PITCH;"~";
1160 SP=PITCH*2-1
     :POSITION 6,9
     :? #6;"NOTE=";N$(SP,SP+1)
     :RETURN
1170 REM [illegible]
1180 FOR OCTAVE=7 TO 1 STEP -1
     :FOR PITCH=12 TO 1 STEP -1
1190 READ FREQ
     :FREQ(OCTAVE,PITCH)=FREQ
     :NEXT PITCH
     :NEXT OCTAVE
1200 BOTH=0
     :LOWOCT=7
     :WAIT=10
1210 REM [illegible]
1220 FOR OCTAVE=LOWOCT TO 1 STEP -1
     :FOR PITCH=12 TO 1 STEP -1
1230 SETCOLOR 4,PITCH,0
     :GOSUB 1140
     :VOL=8
     :POKE 540,VOL
1240 IF  NOT BOTH THEN JW=USR(ADR(S16$
     FREQ(OCTAVE,PITCH),VOL)
     :GOTO 1260
1250 JW=USR(ADR(S16$),FREQ(OCTAVE,
     PITCH),VOL,FREQ(OCTAVE+1,PITCH),
     VOL)
1260 IF  NOT VOL THEN 1280
1270 VOL=PEEK(540)
     :GOTO 1240
1280 GOSUB 1440
1290 NEXT PITCH
     :NEXT OCTAVE
1300 IF  NOT BOTH THEN BOTH=1
     :LOWOCT=6
     :GOTO 1220
1310 FOR ME=5 TO 9 STEP 2
     :POSITION 6,ME
     :? #6;"~~~~~~~~~~";
     :NEXT ME
1320 REM [illegible]
1330 VOL=8
     :OCTAVE=5
     :PITCH=7
     :HOLD=16
     :WAIT=8
     :GOSUB 1490
1340 PITCH=12
     :HOLD=4
     :WAIT=0
     :GOSUB 1490
1350 OCTAVE=6
     :PITCH=1
     :HOLD=4
     :WAIT=0
     :GOSUB 1490
1360 OCTAVE=5
     :PITCH=12
     :HOLD=4
     :WAIT=0
     :GOSUB 1490
1370 PITCH=10
     :HOLD=16
     :WAIT=8
     :GOSUB 1490
1380 PITCH=12
     :HOLD=16
     :WAIT=32
     :GOSUB 1490
1390 PITCH=8
     :HOLD=16
     :WAIT=8
     :GOSUB 1490
1400 PITCH=7
     :HOLD=32
     :WAIT=0
     :GOSUB 1490
1410 GRAPHICS 0
     :GOTO 1660
1420 REM [illegible]
1430 REM
     [illegible]
1440 POKE 540,WAIT
1450 IF PEEK(540) THEN 1450
1460 RETURN
1470 REM
     [illegible]
1480 REM [illegible]
1490 POKE 540,HOLD
     :X=USR(ADR(S16$),FREQ(OCTAVE,
     PITCH),VOL,FREQ(OCTAVE+1,
     PITCH),VOL)
1500 IF PEEK(540) THEN 1500
1510 X=USR(ADR(S16$),FREQ(OCTAVE,
     PITCH),0,FREQ(OCTAVE+1,
     PITCH),0)
1520 GOTO 1440
1530 REM [illegible]
1540 DATA 104,201,2,240,33,201,4,
     240,12,170,224,0,240,41
1550 DATA 202,104,104,240,247,
     208,245,104,141,2,210,104,
     141,0
1560 DATA 210,104,104,41,15,9,
     160,141,3,210,104,141,6,210
1570 DATA 104,141,4,210,104,169,
     41,15,9,160,141,7,210,96
1580 REM
     [illegible]
1590 DATA 27357,25821,24372,23003
     21712,20493,19342,18256,
     17231,16264,15351,14489
1600 DATA 13675,12907,12182,
     11498,10852,10243,9668,
     9125,8612,8128,7626,
1610 DATA 6834,6450,6088,5746,
     5423,5118,4830,4559,
     4303,4061,3832,3617
1620 DATA 3414,3222,3040,2869,
     2708,2555,2412,2276,
     2148,2027,1913,1805
1630 DATA 1703,1607,1517,1431,
     1350,1274,1202,1134,
     1070,1010,953,899
1640 DATA 848,800,755,712,672,
     634,598,564,532,501,473,446
1650 DATA 421,397,374,353,332,
     313,295,278,262,247,233,219
```

```
1655 REM [illegible]
1660 POKE 752,1
    :C=RND(0)*16
    :POKE 764,255
1662 SETCOLOR 1,C,0
    :SETCOLOR 2,C,8
    :SETCOLOR 4,C,0
    :POKE 752,1
    :POKE 559,34
1664 F$="*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*"
1665 ?
    :?
    :?
    :?
1667 ? "************************************"
1670 ? "*YOU'VE~JUST~WITNESSED~16~BIT~SOUND*"
1680 ? "*~~~~~WITH~THE~NOTES~DISPLAYED~~~~~*"
1690 ? "*~~~~~~~FOR~YOUR~CONVENIENCE~~~~~~~*"
    :? F$
1700 ? "*~~~~NOW~TO~FURTHER~DEMONSTRATE~~~~*"
1710 ? "*~HOW~THE~COMPUTER~USES~2~REGISTERS*"
1720 ? "*~~~~~~TO~CREATE~16~BIT~SOUND~~~~~~*"
1730 ? "*~~~HERE~IS~YET~ANOTHER~EXAMPLE!~~~*"
    :? F$
    :? F$
1740 ? "*~~~~~PLUG~IN~THE~OLD~JOYSTICK~~~~~*"
1750 ? "*~~~~SO~YOU~CAN~EXPERIMENT~WITH~~~~*"
1760 ? "*~~~~~~~MAKING~16~BIT~SOUND~~~~~~~~*"
1770 ? "************************************"
    :?
    :?
    :?
1775 ? "~~~~~~~[illegible]"
1780 IF PEEK(764)=255 THEN 1780
3000 REM ****************************
3010 REM *                          *
3020 REM *        SOUND16.CP        *
3030 REM *    by Charlie Parker     *
3040 REM *                          *
3050 REM ****************************
3060 REM [illegible]
3070 SOUND 0,0,0,0
    :CH1=0
    :CH2=0
    :POKE 764,255
    :GRAPHICS 2
    :POKE 710,0
    :POKE 752,1
    :? ,"[illegible]"
3072 ? "~~~FOR~CHANNEL~ONE~--~UP/DOWN"
3074 ? "~~~FOR~CHANNEL~TWO~--~LEFT/RIGHT"
3080 ? "~~~~~~[illegible]";
    :POKE 53768,80
    :POKE 53761,160
    :POKE 53763,168
3090 POSITION 3,3
    :? #6;"CHAN1="
    :POSITION 3,5
    :? #6;"CHAN2="
3095 REM [illegible]
3100 IF PEEK(764)<>255 THEN POKE 764,255
    :SOUND 0,0,0,0
    :SOUND 1,0,0,0
    :POKE 559,0
    :RUN "D:SPEAKER"
3110 S=STICK(0)
    :IF S=15 THEN GOTO 3100
3120 IF S=14 THEN CH1=CH1+1
    :IF CH1>255 THEN CH1=0
    :CH2=CH2+1
    :IF CH2>255 THEN CH2=255
3130 IF S=13 THEN CH1=CH1-1
    :IF CH1<0 THEN CH1=255
    :CH2=CH2-1
    :IF CH2<0 THEN CH2=0
3140 IF S=7 THEN CH2=CH2+1
    :IF CH2>255 THEN CH2=255
3150 IF S=11 THEN CH2=CH2-1
    :IF CH2<0 THEN CH2=0
3160 POKE 53762,CH2
    :POKE 53760,CH1
3170 POSITION 9,3
    :? #6;CH1;"~~"
    :POSITION 9,5
    :? #6;CH2;"~~"
    :GOTO 3100
```

SPEAKER

```
1000 REM __Assembly routines
1010 REM
1020 DIM WT$(57)
    :REM __TDELAY.OBJ

1060 REM
1070 DIM BP$(39)
    :REM __Sound speaker

1100 REM
1110 REM __Mainline
1120 DIM B(30)
1130 FOR I=1 TO 30
    :B(I)=0
    :NEXT I
1140 B(9)=1
    :B(12)=10
    :B(15)=30
1150 GOSUB 1830
1160 C=21
    :R=9
1170 GRAPHICS 0
    :POKE 752,1
1180 SETCOLOR 1,0,0
    :SETCOLOR 2,0,10
1190 ? "┌~~~~~~~~~~~~~~~~~~~~┐"
1200 ? "|~~~SPEAKER~SOUND!~~~|"
1210 ? "└~~~~~~~~~~~~~~~~~~~~┘"
1220 ?
1230 ? "THIS~ASSEMBLY~ROUTINE~WILL~
      ALLOW~YOU"
1240 ? "~TO~GENERATE~SINGLE~VOICE~
      SOUNDS~ON~"
1250 ? "~~~~THE~INTERNAL~SPEAKER"
1260 ?
1270 ? "~~~~~~~~~~~~~ ┌~~~~~┐"
1280 ? "REPEAT~~~~~~|~~~~~|"
1290 ? "~~~~~~~~~~~~~└~~~~~┘"
1300 ? "~~~~~~~~~~~~~ ┌~~~~~┐"
1310 ? "DURATION~~~~|~~~~~|"
1320 ? "~~~~~~~~~~~~~└~~~~~┘"
1330 ? "~~~~~~~~~~~~~ ┌~~~~~┐"
1340 ? "SPEAKER~ON~~|~~~~~|"
1350 ? "~~~~~~~~~~~~~└~~~~~┘"
1360 ? "~~~~~~~~~~~~~ ┌~~~~~┐"
1370 ? "SPEAKER~OFF~|~~~~~|"
1380 ? "~~~~~~~~~~~~~└~~~~~┘"
1390 POSITION 2,22
1400 ? "[illegible]~~~~[illegible]";
1410 REM __Play openning tune
1420 FOR I=10 TO 80 STEP 10
1430 BP$(28,28)=CHR$(I)
    :B(18)=I
1440 U=USR(ADR(BP$))
1450 NEXT I
1460 FOR I=90 TO 40 STEP -10
1470 BP$(28,28)=CHR$(I)
    :B(18)=I
1480 U=USR(ADR(BP$))
1490 NEXT I
1500 REM __Display values
1510 FOR I=9 TO 18 STEP 3
1520 POSITION 20-LEN(STR$(B(I))),I
1530 ? B(I)
1540 NEXT I
1550 REM __User input
1560 POKE 764,255
1570 POSITION C,R
1580 ? "€←~PUSH~TRIGGER↓
     ←←←←←←←←←←←STICK~~€↑€↓~€←€→"
1590 S=STICK(0)
    :T=STRIG(0)
1600 K=PEEK(764)
1610 IF K=0 THEN 1880
1620 IF K=47 THEN 1950
1630 IF T=0 THEN 1730
1640 IF S<>13 AND S<>14 THEN 1590
1650 POSITION C,R
1660 ? "~~~~~~~~~~~~~~~↓
     ←←←←←←←←←←~~~~~~~~~~~~~~~"
1670 IF S=14 THEN R=R-3
1680 IF S=13 THEN R=R+3
1690 IF R<9 THEN R=18
1700 IF R>18 THEN R=9
1710 U=USR(ADR(WT$),20)
    :GOTO 1560
1720 REM __Inc or dec value
1730 IF S=7 THEN B(R)=B(R)+10
1740 IF S=11 THEN B(R)=B(R)-10
1750 IF S=13 THEN B(R)=B(R)-1
1760 IF S=14 THEN B(R)=B(R)+1
1770 IF B(R)<0 THEN B(R)=0
1780 POSITION 15,R
    :? "~~~~~~"
1790 POSITION 20-LEN(STR$(B(R))),R
1800 ? B(R)
    :GOSUB 1830
1810 X=USR(ADR(WT$),20)
    :GOTO 1590
1820 REM __Set assy values
1830 BP$(10,10)=CHR$(B(12))
1840 BP$(20,20)=CHR$(B(15))
1850 BP$(28,28)=CHR$(B(18))
1860 RETURN
1870 REM __Playback
1880 POSITION 14,22
    :? "~~~~~~~~~~~"
1890 FOR I=1 TO B(9)
1900 U=USR(ADR(BP$))
1910 U=USR(ADR(WT$),B(18))
1920 NEXT I
1930 POSITION 14,22
    :? "[illegible]"
1940 GOTO 1560
1950 POKE 764,255
    :GRAPHICS 0
1960 RUN "D:THATSALL"
```

NOTES

NOTES

EDUCATIONAL SOFTWARE INC.'s

WONDERFUL WARRANTY

We warrant the magnetic media or cartridge on which the program resides to be free from manufacturing defects and shoddy workmanship under normal operating circumstances. This does not mean you can damage said media and then send it back to us for a refund, nor does it mean that you can use it for another purpose and allow it to be damaged that way.

We are so sure of this product that if you have any problem running it, you can return it to us in its orginial packaging, along with a nice note of explanation, and we will exchange it for another one, free of charge. We do not, however, give refunds.

This wonderful warranty is in effect for a period of thirty (30) days from the date of purchase. All we require is that you fill out the enclosed owner's registration card and return it to us immediately. With your card on file we can keep you informed on what we are up to here in the redwoods.

Thank you,

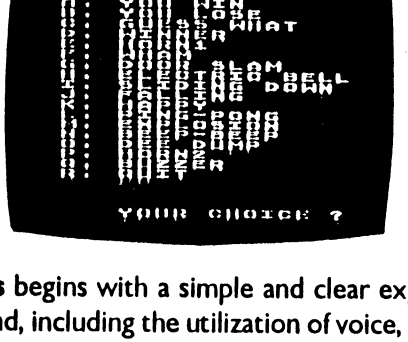
Professor Von Chip

Professor Von Chip

SOUND EFFECTS

by
Jerry White
with
Becky Johnson

From zoo "bleats" to VARROOOOOOMS!!! this Tricky Tutorial™ tells all about making your Atari® produce those special sounds you want for your games and other BASIC programs. Included are instructions and examples for synchronizing your sound effects with graphics on your screen; two menus of over 30 sound effects which you can list and include in your own programs; and a utility to help you write your own FOR/NEXT loops for custom special effects.



Sound Effects begins with a simple and clear explanation of the SOUND command, including the utilization of voice, pitch, distortion and volume variables. Then it helps you use the internal sound registers, enabling you to access the impressive capabilities of all four voices of the Atari. Even 16-bit sound controls are demonstrated and clearly explained.

Appropriate for all age levels of programmers

REQUIRES:
BASIC cartridge
Joystick
16K tape or 32K disk

© 1983 Educational Software Inc.

ATARI is a registered trademark of Atari, Inc. ESI and Tricky Tutorial are trademarks of Educational Software Inc.

IF U > I THEN COLOR
STICK (0) = THIS: IF TH
IF CHR$(251) = Y THEN P